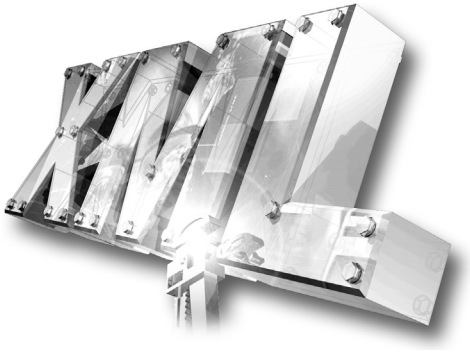
Фрэнк Бумфрей, Оливия Диренцо, Йон Дакетт, Джо Грэф,
Дэйв Холэндер, Пол Хоул, Тревор Дженкинс, Питер Джоунс,
Эдриан Кингсли-Хьюз, Кэти Кингсли-Хьюз, Крэг Маккуин и Стивен Мор

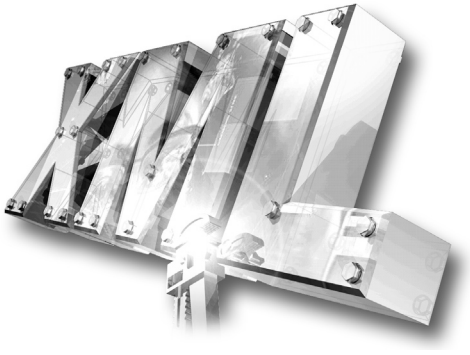
XML
Новые перспективы WWW



XML Applications

Frank Boumphrey,
Olivia Direnzo,
Jon Duckett,
Joe Graf, Paul Houle,
Dave Hollander,
Trevor Jenkins,
Peter Jones,
Adrian Kingsley-Hughes,
Kathy Kingsley-Hughes,
Craig McQueen
and Stephen Mohr

wrox
PROGRAMMERS TO PROGRAMMER™



Серия «Для программистов»

XML
Новые перспективы WWW

Фрэнк Бумфрей,
Оливия Диренцо,
Йон Дакетт,
Джо Грэф,
Дэйв Холэндер,
Пол Хоул,
Тревор Дженкинс,
Питер Джоунс,
Эдриан Кингсли-Хьюз,
Кэти Кингсли-Хьюз,
Крэг Маккуин
и Стивен Мор

ДМК
издательство

Москва

ББК 32.973.26-018.1
Б97

Бумфрей Ф., Диренцо О., Дакетт Й. и др.
Б97 XML. Новые перспективы WWW. Пер. с англ. – М.: ДМК. – 688 с.: ил. (Серия «Для программистов»).

ISBN 5-93700-007-2

В книге в сжатой форме излагаются основы XML – расширяемого языка разметки, а также приводятся примеры его практического использования. На сегодняшний день этот язык считается самым перспективным средством создания Web-документов. Широки его возможности и в качестве средства работы с базами данных и мощного механизма преобразования формата сообщения. Главные достоинства XML – гибкость, свобода в создании самых разнообразных тэгов, способность объединять информацию из различных источников в единый непротиворечивый документ. С языком XML тесно связаны самые новейшие разработки в Web-технологиях, такие как XML-схемы и пространства имен.

ББК 32.973.26-018.1

ISBN 1-861001-9-08 (англ.)
ISBN 5-93700-007-2 (рус.)

Содержание

Об авторах

Фрэнк Бумфрей (Frank Boumphrey) в настоящее время работает в фирме Cormorant Consulting, которая специализируется на создании баз данных в области медицины и юриспруденции. Программированием он начал заниматься еще в древние времена перфокарт и машинных языков. Одним из первых его заданий являлось написание программы, с помощью которой можно было отличить советскую межконтинентальную баллистическую ракету от стаи гусей. Судя по тому, что сейчас мы читаем об этом, программа получилась!

Устав мыслить шестнадцатеричными числами, он оставил программирование и получил диплом доктора медицины. Со временем Фрэнк Бумфрей стал профессором и возглавил отделение хирургии позвоночника в крупном институте на Среднем Западе. В дополнение к своим основным обязанностям Фрэнк занимался внедрением MRI в медицинскую практику. Выйдя на пенсию, он вернулся к своему первому увлечению – компьютерам и теперь пытается убедить медицинские учреждения хотя бы в какой-либо степени рационально организовывать поступающую информацию. С этой целью он читает медицинским работникам и администраторам системы здравоохранения лекции по основам электронного хранения и обработки документации.

Интересно, что наибольший успех Фрэнк Бумфрей имеет у юридических фирм, желающих рационализировать свои базы данных по медицине. Сейчас его главная цель – помочь XML стать основным языком Web-документов.

Оливия Диренцо (Olivia Direnzo) – художница, увлекается растениями и животными. Оливия окончила Корнелльский университет (Cornell) по специальности «зоология». Вместе с Полом Хоулом она работает в компании Honeylocust Media Systems, в которой занимается исследованием гиперсред, основанных на Web. Они оба считают, что это одно из интереснейших в настоящее время направлений.

Йон Дакетт (Jon Duckett), окончив Брюнельский университет (Brunel) в Лондоне и получив степень по психологии, возвратился домой, в Бирмингем, и поступил на работу в представительство издательства Wrox. Благодаря отличным ребятам из издательства Wrox он не слишком скучает по Лондону. Здесь, в Бирмингеме, Джон нашел точку приложения своих сил, стараясь в одиночку поддержать экономику нескольких стран, производящих чай.

Джо Грэф (Joe Graf) в течение шести лет работал консультантом по разработке крупномасштабных клиент-серверных систем, а затем основал компанию по производству программного обеспечения, получившую название E-comm Group

inc. Деятельность компании направлена на развитие языка XML как среды для совершения транзакций в Internet. В последнее время Джо возглавляет разработку и поддержку OFX Script, одного из первых серверов XML-приложений (XML Application Server), который позволит финансовым учреждениям осуществлять денежные транзакции с помощью Internet.

Дэйв Холэндер (Dave Hollander), после окончания Мичиганского технологического университета работал в лаборатории Bell, затем в нескольких компаниях, занимающихся компьютерной графикой, а также в Phillips DuPont Optical. В 1988 году Дэйв поступил в компанию Hewlett-Packard, где возглавил разработку издательских программ, производственных DTD и других систем. Дэйв создал сайт www.hp.com, и за тот период, что он управлял им, число посетителей сайта выросло с тысячи до миллиона в день. В настоящее время Дэйв занимается разработкой основанных на языке XML и предназначенных для связи с помощью каналов систем управления содержанием.

В течение всего этого времени Дэйв являлся образцовым мужем и отцом, возглавлял отряд молодых скаутов, председательствовал в школьном комитете, а также активно участвовал в формальной и неформальной деятельности в области определения стандартов, включая формат Rock Ridge (компакт-диски), Davenport и OSF DTD. Дэйв является действительным членом рабочей группы XML, соавтором спецификации по пространствам имен XML и сопредседателем рабочей группы консорциума W3C по XML-схемам.

Пол Хоул (Paul Houle) недавно получил в университете Cornell степень доктора натуральной философии по физике[1]. В настоящее время работает в Германии, в Институте физики комплексных систем Макса Планка.

Пол отдает предпочтение операционной системе Linux и пишет программы на языках C, Perl и Java.

Тревор Дженкинс (Trevor Jenkins) – независимый консультант по системам управления текстами и документами. В числе его недавних клиентов можно было встретить и сеть ресторанов быстрого обслуживания, и главного издателя STM. До этого ему приходилось выполнять заказы для ведущих компаний в области телекоммуникаций и бухгалтерского дела. Язык SGML для него родной. Тревор участвует в работе BSI и ISO – ассоциаций, занимающихся внедрением SGML и относящихся к этому языку стандартов. В сферу их деятельности также входит техническая оценка языка XML. Тревор Дженкинс участвовал в подготовке нескольких технических отчетов по SGML, которые были опубликованы ведущей в области стандартизации организацией ISO («Методы использования SGML» и «Системы ввода текстовых данных»).

Питер Джоунс (Peter Jones) является техническим редактором и «домашним» автором издательства Wrox. Увлечений у него не счесть, однако с тех пор, как он начал работать в издательстве, времени ему хватает только на то, чтобы изредка погонять по окрестностям на огромном мотоцикле. Он хотел бы поблагодарить свою семью, друзей, всех сотрудников издательства Wrox и всех членов группы

[1] Это примерно соответствует российской степени кандидата физико-математических наук. *(Прим.*

XML-Dev, оказывавших ему поддержку при создании этой книги.

Эдриан Кингсли-Хьюз (Adrian Kingsley-Hughes, awkh@khd.co.uk) является техническим директором Kingsley-Hughes Development Ltd., консультационно-учебной фирмы, специализирующейся на языках для Web и языках визуального программирования. Эдриан основал эту фирму в 1996 году, когда изучал химию в Бангорском университете (Bangor). Кроме того, Эдриан – консультант в области развития Internet и программирования под Windows. Он создает обучающее программное обеспечение для таких далеких областей как химия, астрономия и валлийский язык. В свободное время пишет романы ужасов и изучает игру на диджеридо.

Кэти Кингсли-Хьюз (Kathy Kingsley-Hughes, kkh@khd.co.uk) работает управляющим директором в фирме Kingsley-Hughes Development Ltd. Со времени выхода в свет популярного сетевого журнала-мультфильма «Канал дракона» (The Dragon Channel), написанного на динамическом HTML, она увлеклась каналами CDF. Это издание журнала оказалось одним из первых активных каналов Internet в Великобритании. Перед этим Кэти в течение шести лет являлась инструктором по информационным технологиям, затем начала изучать химию и компьютеры в Бангорском и Открытом университетах. Она интересуется дистанционным обучением и Internet-классами. Кроме того, Кэти преподает химию, физику и математику.

Крэг Маккуин (Craig McQueen) – главный консультант компании Sage Information Consultants Inc. Он специализируется в разработке компонентов COM среднего уровня для электронной коммерции. Участвует в работе над рядом проектов в издательстве Wrox, а также каждый месяц пишет статьи для информационного бюллетеня Visual C++ Developer. В свое время Крэг возглавлял два проекта создания программного обеспечения для Internet: InContext FlashSite (Контекстный флэш-сайт) и InContext WebAnalyzer (Контекстный анализатор Web).

Крэг получил степень магистра в университете Торонто, где изучал взаимодействие человека и компьютера на факультете компьютерных наук. Итогом его исследований могут служить пять публикаций и три выступления на международных конференциях. Адрес электронной почты Крэга – cmcqueen@sageconsultants.com.

Стивен Мор (Stephen Mohr) начал заниматься программированием в школе – в те времена, когда это увлечение было не так популярно, как сейчас. Последние десять лет он специализировался в программировании для персонального компьютера. В качестве старшего архитектора систем программного обеспечения в фирме Omicron Consulting Стивен проектирует и создает системы, используя при этом языки C++, Java, JavaScript, COM, а также различные стандарты и протоколы Internet. Стивен имеет степени бакалавра и магистра компьютерных наук, полученные в политехническом институте Rensselaer. Занимается исследованием распределенных объектно-ориентированных систем и вопросами практического применения искусственного интеллекта.

Предисловие переводчика

В Internet появился новый язык. Зачем? Разве мало нам доброго старого HTML? Да, уже мало. То, что именно HTML способствовал бурному росту Internet, точнее, той его области, которая называется World Wide Web, давно стало общим местом в публикациях, посвященных Internet. Но теперь Сеть переросла своего «родителя» и требует таких возможностей, которых у него нет. Или почти нет.

HTML вполне справлялся с отображением некоторой информации в заранее заданной фиксированной форме. Но развитие Сети поставило новые задачи, для решения которых необходимо разделить данные и способ их представления. Вот простейший, но понятный всем пример. Много раз было сказано, что Internet по содержанию представляет собой сокровищницу, а по способу организации – свалку. Всякий, кто пользовался поисковыми машинами, знаком с проблемами информационного шума и поиска нужных сведений среди результатов поиска. Рутинная задача – найти нужную информацию – требует немалого искусства и затрат времени. Полностью автоматизировать ее не удается. Почему? Именно потому, что HTML не разделяет данные и способ их отображения. Эту задачу решает новый язык – XML.

Язык XML появился совсем недавно и бурно развивается. Эта книга – одно из первых посвященных ему изданий на русском языке и уж почти наверняка первое из них, посвященное практической работе с XML. Как ни лестно идти в первых рядах, при этом возникают свои проблемы, и не последняя из них – перевод терминологии.

Сказать, что русская терминология в области XML не устоялась, значит ничего не сказать. В многочисленных публикациях, преимущественно переводных статьях обзорного характера, появляющихся в русскоязычной компьютерной прессе и русской части Internet, царит полнейший разнобой. Не только один и тот же термин переводится разными словами, это еще полбеды, но бывает и так, что одно и то же слово в разных статьях употребляется для перевода разных терминов. А некоторые переводы слова «entity», одного из ключевых терминов XML – «категория» или «сущность» – заставляют вспомнить лекции по диамату. В такой ситуации остается только уподобиться Шалтаю-Болтаю из «Алисы в Зазеркалье» и провозгласить: «Когда я беру слово, оно значит то, что я хочу». Все же мы старались избегать полного произвола и придерживались терминологии, применяемой может быть и не в самых многочисленных, но в наиболее интересных статьях. Вот краткая сводка переводов некоторых терминов XML, принятых в этой книге:

Well-formed document	(синтаксически) правильный документ
Valid document	состоятельный документ
Parser	анализатор
Validation	верификация
Validating parser	верифицирующий анализатор
Non-validating parser	неверифицирующий анализатор
Entity	компонент
Parameter entity	параметрический компонент
White spaces	пробельные литеры.

Возможно, манера изложения, принятая в этой книге, покажется отечественному читателю чересчур обстоятельной. Но учебный, по существу, характер книги, традиции издательства Wrox Publishers, в котором она выпущена, а также то, что книга посвящается относительно новому предмету, для которого трудно пока определить, какие сведения следует считать известными по умолчанию – все это вполне оправдывает некоторые длинноты и повторы.

Со времени выхода оригинального издания книги прошло достаточно времени, чтобы текущая ситуация, особенно в такой быстро развивающейся области, как язык XML, серьезно изменилась. Например, браузер Internet Explorer 5.0, на момент написания оригинала книги существовавший в виде бета-версии, в настоящее время уже доступен, в том числе и в локализованной русскоязычной версии. Можно только призвать читателя последовать совету, многократно повторяемому авторами книги: следите за новинками и загружайте обновленные версии приведенных кодов с сайта издательства Wrox, адрес которого неоднократно встречается на ее страницах.

Хотелось бы завершить это краткое вступление небольшим перечнем ссылок на ресурсы русского Internet, посвященные языку XML. Увы, пока это невозможно. Большая часть публикаций по XML в отечественной части Internet представляет собой переводы с английского статей обзорного характера или коллекции ссылок на англоязычные, по преимуществу, Internet-ресурсы (например, http://citforum.indi.ru/internet/xml/links.shtml). Особое место среди этих материалов занимает основательная и хорошо написанная статья Александра Печерского «Язык XML – Практическое введение». (Часть первая: http://citforum.syzran.ru/internet/xml/index.shtml. Часть вторая: http://www.citforum.bonus.ru/internet/xml2/index.shtml). Ее смело можно порекомендовать отечественному читателю для первого (и довольно основательного) знакомства с предметом.

Язык XML быстро развивается и со временем, вероятно, распространится так же широко, как и его предшественник HTML, хотя и навряд ли вытеснит его полностью (а почему – вы поймете, прочитав эту книгу). Помимо всех перечисленных в первой главе этой книги причин, по которым стоит изучать XML, есть еще одна – это очень интересно.

Виталий Глинка

Введение в XML

В наше время наиболее широко известной и впечатляющей новаторской разработкой в области Web-технологий можно считать XML, или расширяемый язык разметки (Extensible Markup Language). Надеемся, что, прочитав нашу книгу, читатели согласятся с этим утверждением. Те из них, кто уже освоил язык HTML, будут приятно удивлены гибкостью XML. Овладев этим способом разметки, вы не просто сэкономите время на создании приложений; вас несомненно поразит широта возможностей, которые XML предлагает вам как Web-разработчику.

Цель этой книги – подробно рассказать о языке XML и разнообразных методах, которые необходимо изучить для создания XML-приложений. Затем, на примере действующих XML-приложений, мы рассмотрим, как этот язык взаимодействует с некоторыми из новейших разработок Internet-технологий, такими как ActiveX Data Objects (объекты данных ActiveX, ADO) и Active Server Pages (активные серверные страницы).

Для кого написана эта книга

Настоящий сборник предполагает наличие у читателя базовых знаний по World Wide Web, а также знакомство с языком HTML, при этом читателю совсем необязательно иметь представление о языке XML. После изложения основ, необходимых для понимания XML и способов его использования, мы перейдем к конкретными задачами и постараемся научить вас создавать мощные XML-приложения. Последние главы отведены материалам, в которых используются скрипты Active Server Pages и SQL, языки Java и C++. Овладение подобными технологиями очень полезно для понимания некоторых примеров из последних глав, однако с архитектурой XML-приложений вы сможете разобраться, даже если не вполне владеете подобными новейшими приемами.

Эту книгу вам следует прочесть в том случае, если:

- вы хотите как можно больше узнать о языке XML;
- вам постоянно требуется больше тэгов, чем может предоставить HTML;
- вы хотите создавать свои собственные тэги;
- вы хотите, чтобы ваша разметка несла какой-то смысл;
- вы хотите разделить стиль и содержание документа;
- вы нуждаетесь в более мощном способе создания Web-приложений;
- вы не хотите остаться за бортом, когда в постоянно меняющемся мире Web начинается новая революция.

Что необходимо для работы с книгой

К созданию вашего первого XML-приложения вы можете приступать хоть сегодня. Для этого существует множество достаточно изощренных редакторов HTML- и XML-файлов, с помощью которых можно взяться за это дело, однако на первый раз вам вполне подойдет простейший текстовый редактор, например Windows Notepad (Блокнот). Если у вас есть компьютер, браузер типа Netscape Communicator 4.x или Internet Explorer 4.x, а также доступ к Internet для загрузки компонентов, необходимых при работе над примерами, сразу можете начинать писать на языке XML.

Прежде чем погрузиться в изучение языка XML, полезно взглянуть на саму разметку и историю ее развития в контексте языков SGML и HTML.

Что такое язык разметки

С разметкой, сами того не осознавая, мы сталкиваемся каждый день. Любой элемент документа, который придает тексту особый смысл или обеспечивает дополнительной информацией, можно отнести к разметке. Например, выделение текста является одним из способов привлечения внимания к тому или иному месту в документе.

Но подобное оформление текста не имеет никакого значения, если никто другой его не понимает. Таким образом, для пользования элементами разметки необходим набор правил, в которых надо:

- сформулировать, что именно является разметкой;
- точно описывать средства, с помощью которых она выполняется.

Всякий подобный набор называется языком разметки. Она бывает стилистическая, структурная и семантическая.

Стилистическая разметка

Этот вид показывает, как будет выглядеть документ. Примером стилистической разметки является использование в текстовом редакторе жирного шрифта или курсива. В языке HTML к стилистической разметке относятся тэги `<FONT>`, `<I>`, `<B>`.

Структурная разметка

Этот вид информирует о структуре документа. Тэги `<Hn>`, `<P>` и `<DIV>` являются примерами структурной разметки, которая указывает соответственно на заголовок, абзац и секцию-контейнер.

Семантическая разметка

Этот вид сигнализирует о содержании данных. Примерами семантической разметки являются тэги `<TITLE>` и `<CODE>` .

Языки разметки (markup languages) определяют правила, с помощью которых как раз и придается значение структуре и содержанию документов. Они описываются грамматикой и синтаксисом, в соответствии с которыми следует «говорить» на этом языке.

Тэги и элементы

Многие из тех, кто знаком с языком HTML, часто путают значения понятий *тэги* (tags) и *элементы* (elements). Тэги – это угловые скобки и текст между ними. Вот некоторые примеры использования тэгов в языке HTML:

`<P>` – тэг, отмечающий начало нового абзаца;

`<I>` – тэг, указывающий на то, что следующий за ним текст должен быть выведен курсивом;

`</I>` – тэг, указывающий конец части текста, которая должна быть выведена курсивом.

Под элементами же обычно понимают тэги в совокупности с их содержанием. Следующая конструкция является примером элемента:

```
<B>Это текст, выделенный жирным шрифтом</B>
```

В обычных терминах тэги представляют собой значки, которые предписывают агенту пользователя (например, браузеру), как поступить с тем, что заключено в тэгах.

Уточнение *Агентом пользователя, или просто агентом (user agent), называется то, что действует от имени пользователя. Ваш начальник считает вас своим агентом; ваш компьютер является вашим агентом; ваш браузер является агентом вашего компьютера.*

Пустые элементы (не имеющие закрывающих тэгов), такие, например, как тэг `<IMG>` в языке HTML, – в языке XML рассматриваются несколько иначе. Однако к этому вопросу мы вернемся позже.

На схеме, приведенной ниже (рис. 0.1), изображены части, из которых состоит элемент:

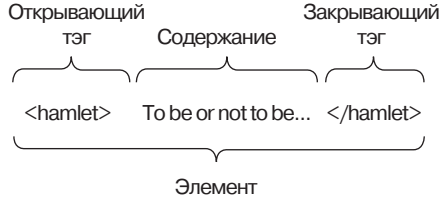


Рис. 0.1. Составные части элемента XML

Атрибуты

Любой тэг может иметь атрибут, если этот атрибут определен. Атрибуты принимают форму пар *имя/значение* (name/value) (их также называют парами *атрибут/значение* – attribute/value). Таким образом, каждому элементу может быть присвоен атрибут с именем. В то же время этот атрибут должен иметь некоторое значение. Тогда тэги принимают форму:

```
<tagname attribute="value"> (<имя_тэга атрибут="значение">)
```

Например, в языке HTML 4.0 у элемента `<BODY>` могут быть следующие атрибуты:

```
CLASS          ID             DIR            LANG           STYLE          TITLE
BACKGROUND     BGCOLOR        ALINK          LINK           VLINK          TEXT
```

Таким образом, можно записать:

```
<BODY BGCOLOR="#000000" ALINK="#999999" LINK="990099" VLINK=#"888888" TEXT="#999999">
```

Языки разметки

В этом разделе мы рассмотрим три языка разметки: SGML, HTML и, конечно, XML. *SGML* – это метаязык, который используется для создания других языков разметки. Наиболее известным из них, написанным в стандарте SGML, является всеми нами любимый HTML, ведь именно он применяется в Web. Поскольку язык HTML разработан в соответствии с правилами SGML, его называют *приложением SGML*. Проблема практического употребления SGML состоит в том, что он очень сложен. Поэтому нам и интересен XML, созданный специально для Web как упрощенная версия SGML, сохраняющая большую часть его функциональных возможностей. Рассмотрим подробнее возможности каждого из упомянутых выше языков разметки.

Язык SGML

В 1986 году, задолго до того, как идея создания Web была воплощена в жизнь, *универсальный стандартизированный язык разметки* SGML (Standardized Generalized Markup Language) был утвержден в качестве международного стандарта (ISO 8879) определения языков разметки. Кстати, SGML существовал еще с конца шестидесятых. В ту пору его использовали для того, чтобы описывать языки разметки, при этом автору позволялось давать формальные определения каждому элементу и атрибуту языка. Таким образом, программисты имели возможность создавать свои собственные тэги, связанные с содержанием документа. В то время SGML был всего лишь одним из нескольких конкурирующих между собой подобных языков, однако популярность одного из его потомков – HTML – дала SGML неоспоримое преимущество перед своими собратьями.

Как язык SGML является очень мощным средством. Однако вместе с мощью пришла и сложность, поэтому все его широкие возможности используются редко. Кроме того, SGML-документ трудно интерпретировать без определения языка разметки, которое хранится в *определении типа документа* (Document Type Definition, DTD). В DTD сгруппированы все правила языка в стандарте SGML. DTD необходимо посылать вместе с SGML-документом или включать в документ для того, чтобы можно было распознать тэги, созданные пользователем. Языки разметки, созданные в стандарте SGML, известны как *SGML-приложения*.

Язык HTML

Первоначально язык HTML был всего лишь одним из SGML-приложений. Он описывал правила, по которым должна быть подготовлена информация для World Wide Web. Таким образом, язык HTML – это набор предписаний SGML, сфор-

мулированных в виде определения типа документа (DTD), которые объясняют, что именно обозначают тэги и элементы. В случае языка HTML, DTD хранится в браузере, оно описано во множестве книг, а также на нескольких Web-сайтах. По размеру язык HTML во много раз меньше языка SGML. В то же время он намного проще и легок для изучения, что добавило ему популярности. Сейчас HTML принят во всех кругах компьютерного сообщества.

Язык HTML как способ разметки технических документов был создан Тимом Бернерсом-Ли (Tim Berners-Lee) в 1991 году специально для научного сообщества. С его помощью оказалось возможным значительно упростить организацию специальных текстов и передачу их через компьютеры различного типа. Идея состояла в создании набора особых словесных формул, которые можно было употреблять для разметки документов. Применение подобных формул должно было обеспечить передачу документов между компьютерами таким образом, чтобы адресаты могли воспроизводить документ в удобном формате. Например:

```
<H1> Это заголовок первого уровня </H1>
<H2> Это заголовок второго уровня </H2>
<PRE> Это текст, для которого важно сохранить форматирование </PRE>
<P> Текст между этими тэгами образует абзац </P>
```

В те далекие времена представители научного сообщества почти не обращали внимания на внешний вид посылаемых и получаемых документов. Ученым было важно сохранить смысл передаваемого текста. Их не волновали такие мелочи, как цвет шрифта или точный размер заголовка первого уровня.

Для передачи информации по Internet язык HTML использует так называемый *протокол передачи гипертекстов* (Hypertext Transfer Protocol, HTTP). Это только один из протоколов, используемых в Internet, входящий в широко известный *набор протоколов Internet* (Internet Protocol Suite), который чаще называют *TCP/IP*. В настоящее время широко используются и несколько других протоколов из набора TCP/IP. До того как появился язык HTML, самым популярным был *протокол передачи файлов* (File Transfer Protocol, FTP).

Преимущество перед другими протоколами дала HTTP легкость, с которой он мог быть использован для подключения к другому документу. Объединение этого протокола с простым для изучения языком обеспечило быстрое распространение систем, реализующих язык HTML и протокол HTTP.

Однако по мере того, как HTML приобретал все более широкое распространение и Web-браузеры становились все доступней, пользователи, не входившие в научное сообщество, стали в массовом порядке создавать свои собственные страницы. Эти представители «ненаучных» кругов все чаще начали обращать внимание на внешний вид своих материалов. Производители браузеров, используемых для просмотра Web-сайтов, с готовностью предлагали различные тэги, которые позволяли авторам Web-страниц представлять свои документы в куда более разнообразном виде, чем просто текст ASCII. Первой на этот путь встала компания Netscape, которая добавила знакомый нам тэг `<FONT>`, позволявший пользователям менять как сами шрифты, так и их размер и ширину. С этого начался быстрый рост числа тэгов, поддерживаемых браузерами.

С новыми тэгами пришли новые проблемы. Различные браузеры воспроизводили новые тэги по-разному. Сегодня существуют Web-сайты, на которых специально указывается: «Лучше всего просматривать в Netscape Navigator» или «Сделано для Internet Explorer», и при всем этом от пользователей ожидают, что созданные ими Web-страницы будут похожи на документы, оформленные в самых совершенных настольных издательских системах.

Таким образом, потенциал браузера как новой платформы для приложений был признан очень быстро, и Web-разработчики приступили к созданию распределенных прикладных систем для бизнеса, используя Internet в качестве среды для получения информации и осуществления финансовых транзакций.

Недостатки языка HTML

В связи с широким распространением языка HTML все больший круг пользователей вовлекался в процесс написания HTML-приложений. Их усилия в первую очередь были направлены на увеличение числа и сложности операций, осуществляемых в Web. В результате скоро стали очевидны слабые места языка HTML, а именно:

- HTML имеет *фиксированный набор тэгов*. Вы не можете создавать свои тэги, понятные другим пользователям;
- HTML – это исключительно *технология представления данных*. HTML не несет информации о значении содержания, заключенного в тэгах;
- HTML – *«плоский» язык*. Значимость тэгов в нем не определена, поэтому нельзя представить иерархию данных;
- *в качестве платформы для приложений используются браузеры*. HTML не обладает достаточной мощью для создания Web-приложений на том уровне, к которому в настоящее время стремятся Web-разработчики. Например, на языке HTML достаточно сложно написать приложение для профессиональной обработки и поиска документов;
- *большие объемы трафика сети*. Существующие HTML-документы, используемые как приложения, засоряют Internet большими объемами трафика в системах клиент-сервер. Примером может служить пересылка по сети большой группы записей общего характера, в то время как необходима только небольшая часть этой информации.

Со временем усилиями пользователей, пытавшихся представить информацию в самых разнообразных формах, Web становилась все более и более фрагментированной. Создавая свои страницы, авторы пытались использовать не только разные реализации языка HTML, но и привлекали языки скриптов, *динамический HTML* (Dynamic HTML), *каналы* (Channels) и другие технологии, которые оказались несовместимы с многими браузерами.

Вот так и случилось, что, с одной стороны, язык HTML являлся очень удобным средством разметки документов для использования в Web, а с другой – документ, размеченный в HTML, нес мало информации о своем содержании, и это в то время, когда использование документа в деловых целях требовало серьезных знаний о его сути. Если тот или иной документ несет достаточно полную информацию о своем содержании, появляется возможность сравнительно легко провести автоматичес-

кую обобщенную обработку и поиск в файле, хранящем документ. Язык SGML позволяет сохранять информацию о содержании документа, однако вследствие особой сложности он никогда не использовался так широко, как HTML. Рассмотрим пример, насколько полезным может оказаться включение в документ информации о содержахся в нем сведениях.

Представьте, что у вас есть библиотека компакт-дисков, размеченная в HTML и хранимая на Web-сервере. Если у вас появилось желание найти фонограмму определенного музыканта, вам придется загрузить библиотеку целиком, а затем произвести поиск по всем записям.

С другой стороны, предположим, что ваша библиотека размечена тэгами, которые содержат информацию о каждом компакт-диске. Например, вы использовали тэг `<artist>` для обозначения имен музыкантов, тэг `<title>` для обозначения названий и так далее. В этом случае вы можете послать запрос серверу только об относящейся к делу части документа, а не обо всем документе. В результате поиск ускоряется, а трафик сети значительно уменьшается.

Задумайтесь вот о чем – если бы вам удалось разметить все документы тэгами, описывающими их содержание, информация о том, как связаться с человеком, могла бы заключаться в тэгах `<firstname>` (имя) и `<email>` (электронная почта), а каталог деталей мог бы использовать такие тэги как `<price>` (цена) и `<partnumber>` (номер детали). Подобные возможности были заложены уже в языке SGML, однако в широкое употребление они так и не вошли. Теперь эта идея нашла свое воплощение в XML.

Как появился XML

Со временем основные производители браузеров дали понять разработчикам, что не собираются в полном объеме поддерживать язык SGML. Более того, его запредельная сложность отпугивала многих людей. Вот почему научная общественность пришла к выводу о необходимости сосредоточить усилия на создании упрощенной версии SGML, предназначенной для использования в Web. Подобное решение открывало перспективу возвращения к документам, размеченным в соответствии со своим содержанием. Консорциум World Wide Web (W3C) первым оценил все преимущества такого подхода и согласился выделить средства на этот проект. Группа экспертов по языку SGML, возглавляемая Джоном Боузэком (Jon Bosak) из компании Sun Microsystems, приступила к работе по созданию подмножества языка SGML, которое могло бы быть принято Web-сообществом. Решено было удалить многие несущественные возможности SGML. Перекроенный подобным образом язык назвали XML. Упрощенный вариант оказался значительно более доступным, чем оригинал, его спецификации занимали всего 26 страниц по сравнению с более чем пятьюстами страницами спецификаций SGML.

Что такое XML

Свое название *расширяемый язык разметки* XML (Extensible Markup Language) получил по той причине, что в нем нет фиксированного формата, как в HTML. В то время как язык HTML ограничивается набором твердо закрепленных тэгов, пользователи XML могут создавать свои собственные тэги, соответствующие содержанию их документа (или применять тэги, созданные другими авторами). Таким образом, XML – это метаязык, то есть особая конфигурация,

с помощью которой можно описывать другие языки. Давайте вернемся к примеру с библиотекой компакт-дисков. С помощью XML мы можем разметить запись о каждом компакт-диске.

```
<cd>
   <artist>Arnold Schwarzenegger</artist>
   <title>I'll Be Bach</title>
   <format>album</format>
   <description>Arnie plays Bach's Brandenburg Concertos 1-3 on the Hammond Organ
   </description>
</cd>
```

В этом примере тэги несут определенную информацию о своем содержании. Тэг `<artist>` сообщает, что его содержание имеет отношение к музыканту, тэг `<format>` говорит, что его содержание описывает формат компакт-диска. Поэтому данный XML-файл, размеченный тэгами, описывающими свое содержание, на самом деле несет информацию и о компакт-диске.

С другой стороны, если вы пожелаете создать документ, включающий информацию о клиентах, вы можете записать все необходимые сведения в файл, приведенный ниже. Обратите внимание на то, как тэги описывают свое содержание:

```
<customer id="1023">
   <company-name>Groovydesign</company-name>
   <first-name>Alex</first-name>
   <last-name>Homer</last-name>
       <house-number>10</house-number>
       <street-name>North Greenside Avenue</street-name>
       <town>Sunnytown</town>
       <state>CA</state>
       <zip-code>94026</zip-code>
       <tel-no>408-725-0975</tel-no>
       <fax-no>408-725-0976</fax-no>
       <email>alex@groovydesign.com</email>
   <on-order>Professional ASP Techniques for Webmasters</on-order>
       <qty>3</qty>
       <date-ordered>09-11-98</date-ordered>
       <delivery-date>09-14-98</delivery-date>
   <US$balance>-149.97</US$balance>
       <previous-purchases>
           <purchase qty=2>Pro ASP 2.0</purchase>
           <purchase qty=5>Instant HTML 2nd edit</purchase>
           <purchase qty=3>Pro MTS MSMQ with VB and ASP</purchase>
       </previous-purchase>
</customer>
```

По сравнению с предыдущим документом, этот файл содержит намного больше сведений, и несмотря на то, что эта запись вам никогда ранее не встречалась, вы вполне можете понять содержание каталога. Если бы такие записи существовали для каждого клиента, их можно было бы использовать:

- как список адресов клиентов;

- как источник информации о величине задолженности для бухгалтерии;
- как источник информации для отдела маркетинга о направленности интересов клиентов;
- для вычисления объема продаж по каждому виду товара.

Совет *Исключительно важно, что язык XML, в отличие от языка HTML, распознает заглавные и строчные буквы, поэтому <artist> <Artist> и <ARTIST> являются разными тэгами.*

Но как же узнать, о чем именно говорят эти тэги и как следует отображать их содержимое? Если вы рассчитывали ограничиться изучением только *одного* нового языка, с помощью которого можно совершить революцию в Web, вас ожидает глубокое разочарование. Создание XML-приложений – процесс куда более сложный, чем программирование на одном, отдельно взятом языке. В XML содержится много разнородных составляющих, из которых надо сложить стройную, цельную картину. Это своего рода игра в мозаику.

Теперь самое время остановиться на кратком описании каждой из этих долей, которые включают:

- определение типа документа (DTD);
- таблицы стилей;
- расширяемый язык создания ссылок;
- просмотр XML;
- анализаторы.

Кроме того, словно этого недостаточно, необходимо создавать новые технологии для совместного использования с XML. Они включают пространства имен (Namespaces), XML-данные (XML-Data) и определения содержания документа (Document Content Definitions). Мы вернемся к ним в следующей главе, а сейчас давайте рассмотрим основы.

Определение типа документа

Идея создания собственных тэгов, имеющих специальное значение и помогающих объяснить содержание документа, сама по себе просто замечательна. Но если каждый пользователь начнет с упоением изобретать все новые и новые описания, каким образом их распознавать? С этой целью в спецификации XML для описания подобных «самодеятельных» тэгов используется так называемые *определения типа документа* (DTD). Как уже упоминалось, в том случае, когда мы решили приступить к созданию нового приложения метаязыка, необходимо одновременно выполнить следующие условия:

- описать, что именно является разметкой;
- описать точно, что означает разметка.

Таким образом, в DTD необходимо занести информацию о том, что означают тэги конкретных элементов. В примере с библиотекой компакт-дисков необходимо описать, что означают тэги `<cd>`, `<title>`, `<format>`, `<artist>` и `<description>`. Атрибуты, которые могут быть присвоены элементам, также должны быть описаны в DTD.

Таблицы стилей

Ни в XML-документе, ни в DTD не содержится никаких сведений о том, каким именно образом документ должен воспроизводиться. Для этой цели вместе с документом мы можем использовать *таблицу стилей* (Style Sheet), управляющую его воспроизведением. Таблицы стилей содержат правила, устанавливающие, каков должен быть внешний вид документа. Они могут быть написаны на разных языках, включая *расширяемый язык таблиц стилей* (Extensible Stylesheet Language, XSL), а также механизм *каскадных таблиц стилей* (Cascading Style Sheets, CSS), которые тоже будут рассмотрены в этой книге. Таблица стилей используется в том случае, когда данные необходимо представить в форме, удобной для восприятия человеком, будь то экран, бумага, система чтения для слепых или звуковой агент пользователя. Таблица стилей сообщает агенту, как воспроизвести информацию, заключенную в тэгах. Поскольку XML-документ и таблица стилей хранятся раздельно, в разных ситуациях можно использовать различные таблицы стилей для воспроизведения одного и того же XML-файла. Можно также использовать одну и ту же таблицу стилей для воспроизведения нескольких XML-документов в аналогичном формате. Таблицы стилей будут детально рассмотрены в главах 7 и 8.

Просмотр XML

Поскольку язык XML относится к области новейших технологий, текущая четвертая версия двух основных браузеров не поддерживает просмотр XML-файлов в полном объеме. Однако это не означает, что ваши XML- и XSL/CSS-файлы уже сейчас нельзя использовать в Web. Для этого придется всего лишь преобразовать файл XML и его таблицу стилей в HTML (или любой другой формат, доступный для просмотра) таким образом, чтобы файл-источник XML мог быть интерпретирован браузером. Существует несколько способов, с помощью которых можно произвести эту операцию, большинство из них будут рассмотрены в различных разделах нашей книги. К счастью, похоже, что скоро нам уже не понадобится заниматься подобным преобразованием, поскольку как Netscape, так и Microsoft планируют в пятых версиях браузеров Communicator и Internet Explorer ввести значительно расширенную поддержку языка XML.

Анализаторы

Использование стандарта XML требует обязательного наличия двух компонентов:

- XML-процессора;
- приложения.

XML-процессор (XML processor) применяется для проверки на соответствие XML-файла спецификации XML. Чтобы компьютер мог интерпретировать XML-файлы, XML-процессор создает конструкцию, известную под названием *дерево документа* (document tree). Именно этим деревом пользуется компьютер, чтобы точно следовать инструкциям процессора. Роль XML-процессора играет *анализатор* (parser). Затем *приложение* (application) обрабатывает данные, содержащиеся в дереве.

Кроме того, анализаторы используются также в качестве инструмента, прове-

ряющего синтаксис и структуру документа. По мере развития HTML основные производители браузеров выпускали все новые и новые версии своих программ. В конечный продукт они включали все больший и больший объем кода, который позволял браузеру воспроизводить HTML-страницы, не только не в полной мере соответствующие спецификациям HTML, но и вообще неправильно написанные. Предполагается, что документы на языке XML будут создаваться более тщательно, и анализаторы станут проверять их соответствие правилам XML.

Создание ссылок

Простые ссылки, которые применяются в языке HTML, обеспечили возможность свободного перемещения как внутри Web-сайтов, оформленных в HTML, так и между ними. Кроме того, они существенно способствовали развитию практики использования Web как среды гипертекстов. Однако механизм создания ссылок в HTML ограничен, и некоторые разработчики попытались найти новый, более мощный способ установления связи со своими документами. В действительности XML не меняет способ, с помощью которого ссылки обеспечиваются на базовом уровне; основная структура ссылок языка HTML сохранена. Однако разработчики языка XML сумели создать эффектное дополнение к этой структуре. Спецификация образования ссылок в XML состоит из двух частей: *X-ссылки* (XLinks) и *X-указатели* (XPointers). В спецификации X-ссылок определено, что ссылки могут быть использованы для связей между документами по типу «один ко многим» и «многие к одному», в то время как X-указатели обеспечивают подсоединение к конкретным частям документов. В главе 5 мы подробнее расскажем о создании ссылок в XML.

Достоинства XML

Язык XML отличается исключительным разнообразием, и главная причина его гибкости заключается в том, что разметка XML-файла позволяет описывать его содержание. В этом смысле он не похож на HTML, который является не более, чем приложением для воспроизведения содержания. XML-документ способен нести информацию о включенном в него материале. Это становится возможным по двум причинам. Во-первых, авторы XML-документов могут создавать свои собственные тэги, относящиеся к содержанию документа. Во-вторых, XML несет информацию только о структуре и смысле документа, оставляя форматирование элементов таблице стилей. Таким образом, тэги XML не просто сами что-то обозначают, но и сообщают информацию о содержании элементов, позволяя процессору обрабатывать XML-файл. Помимо этого, сведения о содержании файла могут быть затем повторно использованы на различных компьютерах и в различных приложениях.

Язык XML в качестве данных

Язык SGML первоначально создавался для разметки документов и до сих пор широко используется для этой цели в полиграфической промышленности. Язык HTML, в свою очередь, давно стал общепринятым форматом для разметки документов и передачи их в Web. Конечно, с одной стороны, XML можно рассматривать в том же качестве, что HTML и SGML, поскольку в основном он предназначен для

разметки документов. В то же время языком XML все шире интересуются в других областях, и порой уже совсем в другом качестве. Поскольку разметка XML по сути дела в какой-то мере отражает содержание документа, его можно использовать и как универсальный формат в любых приложениях. Таким образом, XML-файл не только будет воспроизведен на вашем браузере, но, поскольку XML интегрирован в ряд других приложений, с его помощью можно предоставить пользователю данные и для других целей. Например, XML, как язык разметки документов, приобретает все большую популярность в качестве формата хранения различных материалов. В десятой главе мы рассмотрим приложение, которое обеспечивает сохранение технических документов на сервере. Трудность, однако, состоит в том, что тэги тоже должны где-то храниться, поэтому XML не всегда удобен для хранения больших групп записей. В таком случае разработчики обычно предпочитают использовать традиционную базу данных и по мере необходимости преобразовывать ее содержимое в XML. В главе 9 приводится приложение, которое обеспечивает сохранение сведений в базе данных SQL, при этом по мере необходимости оно создает XML-файлы, используя упомянутый выше процесс преобразования.

Таким образом, реальна перспектива возрастания популярности языка XML и освоения его в ряде других приложений. По крайней мере, авторы считают это вполне вероятным. Бухгалтерия на основе электронной таблицы сможет постоянно воспроизводить итоги торгового оборота во внутренней сети (intranet) предприятия. База данных клиентов, написанная на XML? позволит программе электронной почты использовать такую таблицу в случае, когда понадобится одновременно отправлять большое количество материала и проверять платежи по индивидуальным заказам. И это при том, что весь файл, в котором содержится группа подобных сведений, может быть прочитан не только компьютером, но и человеком.

Слияние многих документов

Другое важное достоинство XML заключается в его способности объединять несколько XML-документов в один большой документ. Если, например, ваша библиотека компакт-дисков содержит сведения об ассортименте товаров в многочисленных розничных торговых точках сети, вам вполне по силам составить отчет об ассортименте на региональном и общенациональном уровне. Если применить метод слияния множества документов и при этом воспользоваться возможностью преобразовать содержимое базы данных в XML, нетрудно создать один – итоговый – XML-документ, включающий данные из различных источников.

Взаимодействие с машиной

Как известно, не только люди с помощью браузеров просматривают XML-страницы в Web. Авторы, пишущие на языке HTML, уже, должно быть, привыкли к использованию тэгов `<META>` с ключевыми словами, описывающими содержание Web-сайтов. С их помощью короткие программы, посылаемые поисковыми машинами, адекватно помещают авторские сайты в указатели. Поскольку XML-файлы несут информацию о своем содержании, машинные пользовательские агенты способны обрабатывать информацию, помещенную в файл. Это означает, что в частном случае применения поисковых машин они обеспечивают значительно

более точные результаты по запросам. В то время как HTML стал форматом представления, XML действует в качестве общепринятого синтаксиса, позволяя значительно большему числу машинных пользовательских агентов использовать хранимые в XML-файлах данные для различных целей. Однако XML представляет собой только часть движения по пути описания содержания документов. По мере роста Internet растет необходимость использования метаданных.

Метаданные

Существует мнение, что в настоящее время наши Web-страницы в основном воспроизводятся браузером, который визуально демонстрирует их содержимое на мониторе компьютера. В то время как механизмы типа каскадных таблиц стилей (CSS) позволяют выводить содержимое файлов на устройства для чтения слепыми или на звуковые пользовательские агенты, разработчики прикладывают усилия, чтобы и машина сумела прочитать представляемые данные. За этой идеей просматривается обоснованная надежда на то, что Internet вскоре будет в значительной мере населен машинными пользовательскими агентами, которые станут действовать от имени людей. Например, появятся специальные программы, которые с помощью языка XML смогут найти нужную книгу или самую дешевую копию компакт-диска Арни Шварценеггера, играющего Баха на органе Hammond. Именно *метаданные* (metadata) обеспечивают этот тип обмена информацией, описывающей содержание документов. Таким образом, метаданные являются по сути данными о данных, или, в нашем случае, данными, описывающими ресурсы Web. Это, в свою очередь и наряду с другими возможностями, позволит машинам осуществлять значительно более точный поиск. Другие области применения, на которые особенно активно повлияют метаданные, включают в себя:

- оценку содержания документов;
- описание наборов страниц, представляющих собой единый логический «документ»;
- описание прав интеллектуальной собственности на те или иные страницы;
- выражение частных предпочтений по поводу различных документов и сайтов.

Заинтересованность консорциума W3C в развитии метаданных как раз и инициировала разработку *формата описания ресурсов* (Resource Description Framework, RDF), по существу являющийся языком представления метаданных. Те же причины привели к созданию *платформы отбора содержания в Internet* (Platform for Internet Content Selection, PICS), о которой мы часто слышим в Web при обсуждении методов оценки содержания.

RDF представляет собой язык, выбранный консорциумом W3C для описания метаданных, которые смогут легко обрабатываться машинами. Ожидают, что этот метод будет играть большую роль в автоматизации многих задач.

Формат описания ресурсов

Формат описания ресурсов (RDF) следует рассматривать как основу обработки метаданных, обеспечивающую независимый от приложений обмен информа-

цией, понимаемой машинами. RDF упрощает автоматическую обработку ресурсов в Web.

Существуют две главные области использования RDF:

- модель данных RDF;
- синтаксис RDF.

Модель данных RDF представляет собой абстрактную концептуальную схему представления данных. Это нейтральный к синтаксису способ представления выражений RDF. Представление модели данных часто применяют для оценки эквивалентности значений. Модели данных подвергают сравнению, и если их представления совпадают, то и выражения RDF можно считать эквивалентными.

Синтаксис RDF – это особая структура для создания метаданных и обмена ими. В настоящее время в этом качестве предлагается синтаксис XML, хотя со временем могут появиться и другие варианты. Чтобы установить точное соответствие между каждым свойством и областью, в которой оно определено, RDF также требует использования пространства имен XML (с которым мы познакомимся в главе 4).

Таким образом, RDF и XML дополняют друг друга.

RDF позволяет авторам выбирать словарь по желанию; им лишь нужно знать, как разрешается пользоваться данным словарем. Представьте, что приложение, упорядочивающее компакт-диски, размечено тем самым образом, как и в ранее приведенном примере. Затем вообразите такое же приложение, но только созданное для коллекции живописных работ. Понятно, что словарь будет зависеть от назначения приложения. Соответственно оба приложения могут иметь тэг `<artist>`, однако в отличие от тэга `<format>`, который встретился нам в примере с библиотекой компакт-дисков, в коллекции картин может появиться тэг `<medium>`, сообщающий, написана ли картина акварелью, маслом или пастелью. RDF позволяет понять, какая именно лексика используется в данном случае. Это делается с помощью присвоения Web-адреса каждому отдельному словарю путем указания пространства имен.

Возможность выбирать словарь также означает, что операции, которые мы привыкли выполнять с базами данных – например, такие как запросы, – можно будет производить с данными, размеченными в соответствии с RDF. Вы сможете фильтровать и сортировать затребованную информацию.

Конечно же, следует иметь в виду, что функциональность RDF значительно шире, чем все, сказанное выше. Более подробную информацию вы можете найти по адресу:

http://www.w3.org/Metadata/

Причины, побудившие специалистов заняться разработкой RDF, ясны. По мере расширения Web и увеличения объемов данных, доступ к которым стал реальным с появлением Internet, в той же, а то и более высокой, степени увеличивается потребность в формальном описании информации. Пытаясь с помощью поисковых машин найти те или иные данные, каждый из нас не раз получал наряду с нужными адресами множество адресов, не относящихся к делу. Теперь, когда на сцену выходят метаданные, появляется возможность обработки и поиска инфор-

мации специальными программами-машинами без всякого нашего участия, но от нашего имени. Использование RDF в качестве средства для подготовки анализа поможет эффективнее проводить мета-анализ данных при оценке тренда и показателей бизнеса.

Консорциум World Wide Web

Одним из главных направлений деятельности консорциума World Wide Web (W3C) является оказание содействия при разработке и оформлении стандартов Всемирной Паутины. Финансовую поддержку консорциум получает за счет грантов и членских взносов. Консорциум W3C насчитывает всего около трехсот членов, однако все они чрезвычайно богатые и, что более важно, компетентные люди.

Каждый новый стандарт, впервые представляемый в W3C, сначала становится *предложением* (proposal), а затем, после рассмотрения и одобрения в W3C, *сообщением* (note). Например, сообщение по расширяемому языку таблиц стилей (XSL) было принято в августе 1997 года. Затем это сообщение стало рабочим проектом, который в процессе прохождения по закрытому списку адресатов, включающих членов W3C и приглашенных экспертов, был исправлен и дополнен. После этого проект в августе 1998 года был вновь обнародован; тогда же началась общая дискуссия по открытым спискам адресов рассылки. Любой мог принять участие в этом обсуждении. Некоторые списки адресов принадлежат консорциуму W3C, другие, такие как, например, список DSSSL, находящийся в ArborText, относятся к независимым от W3C исследователям. Все это лишний раз подтверждает, что консорциум W3C прислушивается к общественному мнению.

После обсуждения члены консорциума возвращаются к проекту и рассматривают предложенные рекомендации, по которым голосуют в течение шести недель. В зависимости от результата голосования, проект либо становится официальным стандартом, либо возвращается на стадию проекта.

Компании Microsoft и Netscape и консорциум W3C

Когда же дело, простите за каламбур, доходит до дела, все, что мы написали в этой книге, все протоколы, утвержденные организациями типа W3C, все объявленные международные стандарты ничего не значат, если большинство участников связанной с Internet игры их игнорирует. Факт состоит в том, что девяносто пять процентов пользователей всего мира пользуется браузерами Netscape и Microsoft. Итак, если основные игроки не поддержат язык XML, как быть?

Говоря о языке XML, обратим ваше внимание на то, что дальнейшее совершенствование Web жизненно необходимо для развития промышленности, и поэтому скорее будет подчиняться требованиям рынка в целом, чем рыночным стратегиям основных игроков. Вот почему утверждение, что XML будет существовать и впредь, можно принять за аксиому.

Следующее поколение браузеров будет существенно в большем объеме, чем современное, поддерживать XML с таблицами стилей CSS. Браузер Internet Explorer будет использовать технологию ActiveX для поддержки языка XSL, а его окончательная реализация будет исполmзовать как таблицы стилей XSL, так и CSS. С новейшей информацией по вопросу о поддержке языка XML браузером

Internet Explorer 5 (IE5) можно познакомиться на сайте

http://www.Microsoft.com/XML/

Компания Netscape тоже обещала поддерживать XML, и хотя во время подготовки этой книги еще не было ясности, что будет представлять собой последняя версия ее браузера. Ожидается включение в него языка XML и таблиц стилей CSS, а также поддержка XSL с помощью технологий JavaApplets и JavaScripts. Развитие событий в этой области отражается на Web-странице

http://home.netscape.com/browsers/

В силу приведенных аргументов, свидетельствующих в пользу языка XML, в скором времени наступит момент, когда от наблюдения за происходящим следует переходить к конкретной работе. Мы понимаем, что в наши дни, когда язык XML еще не получил всеобщей поддержки и его стандарты не вполне устоялись, настоятельный призыв заняться изучением языка звучит довольно странно и способен обескуражить кого угодно. Однако учтите, что XML довольно быстро завоевывает признание, и именно теперь пробил час, когда еще не поздно приступить к его изучению. Как было сказано ранее, браузер IE5 будет поддерживать XML, и если окончательная версия к моменту выхода книги еще не появится на свет, вы сможете загрузить бета-версию по адресу:

http://www.microsoft.com/sitebuilder/ie/ieonsbn.htm

В этой книге и на нашем сайте имеются примеры, демонстрирующие XML в действии на бета-версии браузера IE5. Почему бы вам не попытаться поработать с ними? Единственная трудность, связанная с IE5, состоит в запуске приложений, помещенных в главе 10, для которых требуется библиотека DLL браузера IE5, загруженная под управлением IE4. Не стоит беспокоится: в той же главе описано, как справиться с этой загвоздкой. Только учтите, что поскольку это всего лишь бета-версия, в программе могут быть ошибки.

Немного терминологии

Когда нам понадобится точно обозначить вновь созданные в XML тэги, мы будем вести речь о *схемах* (schemas). Вообще, схемы в этом конкретном понимании представляют собой обобщенный план или диаграмму, однако согласно положениям науки о компьютерах схемы, как известно, определяют характеристики классов и объектов. Что же такое *классы* (classes) и *объекты* (objects)?

Все, что мы видим вокруг себя, является объектами. О компьютерах, например, можно мыслить как о классе объектов, называемом `computer`. Тогда мой персональный «дружок», по имени `bobbin`, является примером объекта класса `computer`. И ваш компьютер, и мой компьютер, и компьютер тетушки Мейбл – все они могут служить примерами объектов этого класса. Таким образом, класс – это совокупность объектов с общими свойствами. Выделенная группа свойств объекта определяет, почему тот или иной объект относится к тому или иному классу. С помощью схем выносится суждение, какими свойствами (и как они соотносятся между собой) обладает конкретный класс. Таким образом, схема определяет внутреннюю структуру класса.

Итак, класс указывает параметры, которые следует определить для объекта

в первую очередь. Их соответствие той или иной схеме относит предмет или явление к тому или иному классу. Конечно же, параметры, выбираемые для определения объекта, достаточно субъективны, поскольку разные люди могут выбрать разные критерии для описания одного и того же предмета. Когда вам придется создавать тэги, вам самим придется решать, какие аспекты объекта необходимо в них включить. Несомненно, это будет напрямую связано с задачей, которая стоит перед вами. Наши рассуждения могут показаться немного абстрактными, поэтому давайте займемся конкретным примером – книгами.

Книга вполне может стать классом объектов, который нужно определить. Так и назовем этот класс – `book` (книга). Возможно, вы захотите включить в описание подобных объектов только заглавие и имя автора. Это можно сделать с использованием двух строк, например, `Professional Style Sheets for HTML and XML` («Профессиональные таблицы стилей для HTML и XML») в качестве заголовка и `Frank Boumphrey` в качестве имени автора. Таким образом, книга «Профессиональные таблицы стилей для HTML и XML» является объектом класса `book`, который мы пытаемся описать. Параметры, определяющие объект класса, называются *атрибутами* (attributes). В данном случае атрибутами являются `title` (название) и `author` (автор).

Почему все это необходимо знать? Потому, что схема является определением группы свойств (и их взаимных соотношений) определенного класса объектов. Схемы, с которыми мы встретимся в XML, определяют свойства и структуру класса документов.

Группа XML-Dev

XML-Dev – это список рассылки для XML-разработчиков. В различных главах книги вам часто будут встречаться ссылки на него. Это список в основном предназначен для людей, активно участвующих в создании ресурсов для XML. При этом имеются в виду не только члены рабочей группы (WG), но и те, кто также активно работает над реализацией какой-то из частей спецификации. Одним словом, XML-Dev – это неформальная и немодерируемая группа поддержки тех, кто заинтересован в реализации и развитии XML. Вы можете подписаться на список рассылки, отправив сообщение по адресу:

majordomo@ic.ak.uk

При этом вам нужно только добавить следующие слова:

```
subscribe xml-dev
```

Поскольку этот список может быть очень большим, вам, возможно, покажется более удобным подписаться на сборники тезисов, которые содержат основные темы обсуждения по данному вопросу. Они будут приходить ежедневно, это удобнее, чем получать множество писем, переполняющих почтовый ящик. Для получения тезисов необходимо написать следующее:

```
subscribe xml-dev-digest
```

Архив списка находится на сайте:

http://www.lists.ic.ac.uk/hypermail/xml-dev/

Теперь о порядке обсуждения.

Цитата

Примеры тем для обсуждения:

- *детальная реализация спецификации;*
- *ресурс – например, документация, данные и результаты тестов;*
- *приведение к XML-стандарту таких объектов, как определения типа документа, группы компонентов, каталоги и тому подобное;*
- *API (интерфейсы прикладных программ) для разработчиков программного обеспечения;*
- *проблемы реализации и запросы ресурсов, имеющих отношение к XML;*
- *использование существующих средств языка SGML для создания ресурсов XML.*

Не рекомендуется делать следующее:

- *запрашивать общую информацию о языке XML (обращайтесь к FAQ, то есть к часто задаваемым вопросам);*
- *обсуждать вопросы языка SGML, не относящиеся к XML (обращайтесь к comp.text.sgml);*
- *обсуждать переработку спецификации (обращайтесь к WG).*

Раздел «часто задаваемые вопросы» (FAQ) по языку XML поддерживается Питером Флинном (Peter Flynn) и может быть просмотрен на Web-странице

http://www.ucc.ie/xml/

Подводя первые итоги

Надеемся, вы уже начали осознавать потенциал, заложенный в языке XML. В заключение повторим вкратце некоторые из его достоинств:

- для широкого круга пользователей Internet необходим стандарт языка (SGML недостаточно популярен из-за своей сложности) и скорее всего как раз XML займет это место. Он будет использоваться все шире;
- XML отделяет синтаксис и структуру от представления документа;
- XML дает возможность создавать свои собственные тэги, описывающие содержание документа;
- определяя свой собственный язык разметки, можно кодировать информацию значительно точнее, чем позволяет HTML;
- XML-файл может быть прочитан человеком;
- обрабатывать структурированную информацию смогут не только браузеры. XML обеспечивает расширенную функциональность, предоставляя машинам легкий доступ к файлам;
- XML относительно легко изучить, что позволяет программистам, работающим на HTML, быстро овладеть структурированной разметкой;

- в отличие от языка SGML, для XML определения типа документа не обязательны;
- тэги XML можно применять для управления поиском;
- XML может употребляться в качестве формата обмена для протоколов транзакций;
- XML кодирует не только документы, но и данные;
- XML создан таким образом, чтобы его можно было легко реализовать;
- XML позволяет объединять большое количество файлов и создавать из них составные документы;
- XML обеспечивает работу с различными приложениями, в отличие от HTML, который совместим только с браузерами.

Будьте внимательны

Поскольку технология XML непрерывно и очень быстро развивается, издатели не берутся утверждать, что в нашем сборнике собрана самая последняя информация на эту тему. Столь смелое заявление с нашей стороны было бы неосмотрительным (и, что часто бывает, очень скоро оказалось бы недостоверным). Однако, как подтверждает история, существует момент, когда каждый из нас встает перед выбором: либо сделать решительный шаг и приступить к освоению новой технологии, либо безнадежно отстать. Мы выбрали первое. Чтобы опыт общения с проектом XML оказался по возможности менее болезненным, мы разместили по всей книге информацию о том, куда следует обращаться за разъяснениями, так что читатели вполне могут быть в курсе любых изменений. Более того, поскольку всем нам хочется, чтобы в сутках было больше двадцати четырех часов, мы сделали так, что в разделе `XML Center` (Центр XML) нашего Web-сайта вы можете познакомиться с новостями об основных результатах последних разработок XML.

Адрес нашего Web-сайта:

http://Webdev.wrox.co.uk/XML/

Во время создания сборника мы получали бета-версии браузера Microsoft Internet Explorer 5. С самого начала предполагалось, что, скорее всего, общедоступные компоненты анализаторов окажутся трансформированы в соответствии с модификацией стандартов XML. Например, будут внесены изменения в имена методов и свойств, которые мы использовали в некоторых приложениях, описанных в последних главах. Если это случится, имейте в виду, что теория, на которой основаны эти приложения, останется вполне корректной, однако для работы кода понадобится уточнить некоторые термины и свойства. Любой желающий по мере необходимости может ознакомиться с ними. Изменения к главам будут помещены на нашем Web-сайте, где также будет храниться и весь код, включенный в книгу:

http://webdev.wrox.co.uk/books/1525/

Принятые обозначения

Для того, чтобы помочь читателям свободно ориентироваться в текстах разного вида, для оформления страниц и общего оформления книги мы использовали

ряд стилей. Следующие ниже образцы являются наглядным примером выбранной нами разметки. Здесь же сказано, что эти стили обозначают

Совет *Так выделены советы, подсказки, справочные материалы.*

Уточнение *Так выделены важные сведения.*

Цитата *Так выделены цитаты и прочие сведения из других источников, например, официальных спецификаций.*

Цитаты из спецификаций используются с разрешения консорциума W3C. Авторские права принадлежат World Wide Web Consortium 1995-1998, (Massachusets Institute of Technology, Institut National de Reserche en Informatique et en Automatique, Keio University). Все права защищены:

http://www.w3.org/Consortium/Legal/

Слова, на которые следует обратить внимание, выделены курсивом.

Так же выделены *новые термины.*

Слова, которые появляются в меню, например, **File** или **Window**, напечатаны полужирным шрифтом.

Так же выделяются названия клавиш на клавиатуре, например, **Ctrl** и **Enter**.

Код программы может содержать несколько шрифтов. Если это элемент кода, о котором говорится в тексте, например, цикл `For...Next`, используется шрифт с буквами фиксированной ширины. Если это часть кода, которую вы можете ввести и запустить как программу, она выглядит следующим образом:

```
<SCRIPT>
... Some VBScript ...
</SCRIPT>
```

Иногда вам может встретиться код программы с несколькими стилями, вот такой:

```
<HTML>
<HEAD>
<TITLE>Cascading Style Sheet Example</TITLE>
<STYLE>
style1 {color: red;
   font-size: 25}
</STYLE>
</HEAD>
```

Светлым шрифтом показана та часть кода, которую мы уже рассмотрели и не считаем нужным обсуждать далее.

В тех местах, где ширина страницы не позволяет поместить длинную строку кода, мы часто используем в начале следующей строки символ подчеркивания (_), чтобы указать, что в вашей программе текст кода должен продолжаться на той же строке.

Эти форматы предназначены для того, чтобы вы понимали, о чем идет речь – следовательно, они являются разметкой. Надеемся, эти замечания помогут вам в вашей работе.

Сообщите ваше мнение

Авторы и издательство приложили немало труда, чтобы сделать эту книгу полезной и приятной для чтения. Лучшей наградой для нас служили бы весточки от наших читателей, в которых сообщалось, что сборник им понравился и стоит затраченных денег. Мы сделали все от нас зависящее, чтобы не обмануть ваших ожиданий.

Пожалуйста, поделитесь с нами вашим мнением. Если вас не затруднит, укажите, что понравилось, а что заставило пожалеть о выброшенных на ветер деньгах, заработанных тяжким трудом. Если кто-то из вас полагает, что это всего лишь один из приемов маркетинга, попробуйте черкнуть нам строчку-другую. Не пожалеете!..

Мы ответим и, каково бы ни было ваше послание, примем к сведению все полезное и представляющее интерес и обязательно используем ваши советы при будущих изданиях. Самый простой способ связаться с нами – это электронная почта:

feedback@wrox.com

На нашем Web-сайте вы можете больше узнать и об издательстве Wrox Press. На нем представлены коды из наших последних книг, извещения о готовящихся к выходу изданиях и сведения об авторах и редакторах. Адрес нашего сайта:

http://www.wrox.com

Помощь читателю

Если во время чтения вам встретится ошибка, пожалуйста, сначала просмотрите на нашем Web-сайте список опечаток. Если вы испытываете сомнения, стоит ли добавлять найденную вами опечатку в указанный список, обратитесь к Приложению H, где подробно описана эта процедура.

Полный адрес списка опечаток:

http://www.wrox.com/Scripts/Errata.idc?Code=1525

Если вам не удалось найти ответ на поставленный вопрос, сообщите о возникших трудностях, а мы постараемся быстро связаться с вами. Но, пожалуйста, не забудьте указать название книги, по поводу которой вы посылали запрос, а также, по возможности, издательский номер. Это поможет ускорить ответ.

Сообщения отправляйте по адресу: support@wrox.com.

Глава 1. Складываем мозаику XML

В настоящее время язык XML только-только «выходит в люди», поэтому изучать его – подлинное удовольствие. И это несмотря на то, что нынешний момент не самое лучшее время для такого знакомства. Еще ничего не устоялось, все – терминология, методики, коды, средства поддержки – находится в движении. Не стихает дискуссия по поводу предлагаемых стандартов, постоянно поступают все новые и новые предложения, тем не менее мы надеемся, что сумеем довести вас до цели, пусть даже она порой напоминает движущуюся мишень. В этой главе мы покажем, как собрать воедино различные части XML-мозаики и создать работающее XML-приложение.

Также мы познакомим наших читателей с некоторыми новыми идеями, чтобы каждому из вас стало ясно, за какими направлениями стоит проследить особо. В этой главе мы обсудим:

- определения типа документа (DTD);
- различия между синтаксически правильными (well-formed) и состоятельными (valid) XML-документами;
- таблицы стилей для форматирования XML-документов;
- анализаторы;
- создание ссылок в XML;
- просмотр XML-файла в браузере;
- новые предложения, касающиеся пространств имен (Namespaces);
- новые предложения для XML-схем, включая XML-данные и описание содержания документа (DCD);
- причины, по которым язык XML должен вам понравится.

Биты и части

Как было отмечено во введении, для практического использования языка XML и его просмотра Web-браузером необходим набор частей, которые в совокупности и составляют XML-мозаику. В данной главе приведен обзор этих элементов, с тем чтобы при их детальном рассмотрении в последующих главах вы представляли, как они соотносятся друг с другом. Желая продемонстрировать, что представляют собой отдельные фрагменты XML-мозаики, вернемся к примеру с библиотекой компакт-дисков, который мы уже приводили. Напомним, что полный XML-файл, описывающий ее фонд, должен был бы состоять из множества размеченных аналогично приводимому примеру записей о компакт-дисках, заключенных в корневые тэги `<cdlib>`.

```
<cdlib>
   <cd>
```

```
    <artist>Arnold Schwarzenegger</artist>
    <title>I'll Be Bach</title>
    <format>album</format>
    <description>Arnie plays Bach's Brandenburg Concertos 1-3 on
    the Hammond Organ</description>
    </cd>

    <cd>
        ...
    </cd>

</cdlib>
```

Определение типа документа

Чтобы разметить XML-документ удобными и понятными тэгами, несущими информацию о своем содержании, необходимо установить правила, понятные для языка разметки, а именно:

- описать, что является разметкой;
- точно определить, что означает тот или иной элемент разметки.

Таким образом, на практике мы должны не только подробно описать каждый из элементов, но и порядок их следования, а также указать, какие именно атрибуты они могут принимать. Спецификация языка XML как раз и определяет данные правила, при этом используется *определение типа документа* (Document Type Definition, DTD). Когда DTD посылается вместе с XML-файлом, пользовательский агент вправе ожидать, что документ соответствует приложенному DTD.

Следует, однако, заметить, что поскольку XML продолжает модифицироваться, необходимо иметь представление и о других способах описания посылаемой информации. Все они объединены под названием *XML-схем* (XML schemas). В первую очередь мы рассмотрим определения типа документа, поскольку они являются частью первоначальной и основной спецификации XML 1.0. В конце этой главы мы также остановимся и на других способах представления данных, которые в настоящее время еще находятся на стадии разработки.

Определение типа документа может содержаться внутри *описания типа документа* (Document Type Declaration) или являться внешним файлом

Если DTD является внешним файлом, то оно подсоединяется к документу следующим способом:

```
<!DOCTYPE cdlib SYSTEM "cdlib.dtd">
```

Язык HTML тоже имеет определение типа документа, но оно присутствует в неявной форме, поскольку содержится внутри наиболее распространенных Web-браузеров и поэтому никем не может быть прочитано. С ним можно познакомиться, посетив сайт консорциума W3C: http://www.w3.org/TR/REC-html140/loose.dtd. В соответствии со стандартом HTML, предполагается, что при создании HTML-документа в код должна быть включена следующая строка:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 //EN">
```

Эта запись сообщает агенту, где находится определение типа документа

для HTML-файла. На практике этой строкой часто пренебрегают, поскольку в большинстве случаев в ней нет особой необходимости. Однако в том случае, когда используются тэги какого-либо конкретного браузера, в какой-то мере не соответствующие формальной спецификации, результаты подобного пренебрежения могут оказаться непредсказуемыми.

Работа над созданием вашего личного языка разметки, в котором используется DTD, не представляет больших трудностей. Рассмотрим, например, определение типа документа, описанное внутри XML-файла, относящегося к библиотеке компакт-дисков. Как видите, это достаточно простое DTD. Деталями мы займемся в следующей главе, однако сразу следует сказать, что этого определения достаточно, чтобы пример с библиотекой компакт-дисков работал.

```
<!DOCTYPE cdlib [
<!ELEMENT cdlib (cd+)

<!ELEMENT cd (artist+, title+, format?, description?)>
<!ELEMENT artist (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT format (#PCDATA)>
<!ELEMENT description(#PCDATA)>

]>
```

Короче говоря, строку `<!DOCTYPE cdlib [` используют для того, чтобы отличить документ с описанием библиотеки компакт-дисков от других типов документов. В этой строке указывается имя, совпадающее с именем корневого элемента документа. Строка `<!ELEMENT>` используется для объявления элементов в формате:

```
<!ELEMENT name (contents)>
```

Здесь `name` (имя) соответствует имени элемента, а `contents` (содержание) описывает, какой тип данных может быть использован и какие элементы разрешается вкладывать в данный элемент. Элемент `<cd>` должен включать в себя элементы `<artist>` и `<title>` (на это указывает символ `+`, который означает «один или более»), в то время как элементы `<format>` и `<description>` не обязательны (на это указывает символ `?`).

Затем определяются элементы, включенные в элемент `<cd>`. Они практически могут представлять собой любой обычный текст без разметки (на исключениях мы остановимся в следующей главе).

Уточнение *Очень легко перепутать определение типа документа (DTD) и описание типа документа (тоже DTD). Чтобы не совершить ошибку, помните, что описание типа документа либо ссылается на определение типа документа, находящееся вне документа, либо содержит его в виде объявлений разметки, как в примере, который мы только что продемонстрировали.*

Скорее всего, с развитием XML число применяемых DTD, которые можно загрузить из сети и использовать для описания файла, будет постоянно расти. Следует ожидать, что скоро появятся стандарты, в соответствии с которыми необходимо

будет размечать определенные типы файлов. Этот процесс уже идет – обратите внимание на такие примеры как *формат определения канала* (Channel Definition Format, CDF) и *химический язык разметки* (Chemical Markup Language, CML), которые мы тоже рассмотрим в этой главе. Но даже если вы не собираетесь заниматься описанием собственных определений типа документа, понимание того, как они работают, очень важно для создания XML-приложений.

Правильные и состоятельные документы

Спецификация XML определяет два типа документов: *синтаксически правильные*, или просто *правильные* (well-formed), и *семантически корректные*, или *состоятельные* (valid). Чтобы быть правильным, документ должен соответствовать следующим трем правилам:

- содержать по крайней мере один элемент;
- содержать корневой элемент, который представляет собой уникальный открывающий и закрывающий тэг, заключающий в себя весь документ;
- все другие элементы, находящиеся внутри документа, должны быть вложены друг в друга без перекрытия.

Итак, если мы вновь обратимся к нашему примеру с библиотекой записей, то вот как должен выглядеть правильный XML-документ:

```
<cdlib>
   <cd>
      <artist>Arnold Schwarzenegger</artist>
      <title>I'll Be Bach</title>
      <format>album</format>
      <description>Arnie plays Bach's Brandenburg Concertos
      1-3 on the Hammond Organ</description>
   </cd>
</cdlib>
```

Здесь налицо один обязательный элемент и несколько других. В документе содержится корневой элемент<cdlib>, который можно сравнить с открывающим <HTML> и закрывающим </HTML> тэгами HTML-документов. Его подэлементы, или дочерние элементы, вложены без перекрытия. Таким образом, документ соответствует критерию правильности.

С другой стороны, состоятельные документы обязаны быть не только правильными, но и иметь такое определение типа документа, которое соответствует тем или иным критериям. Это означает, что в таком документе могут быть использованы только те элементы, которые были объявлены в заданном порядке и имеют разрешенные типы содержания, определенные в DTD.

Концепция состоятельного документа заимствована из языка SGML, однако в SGML документы *обязаны* быть состоятельными, поэтому не существует концепции просто правильных SGML-документов. Конкретные вопросы, связанные с правильными и состоятельными документами, будут рассмотрены в второй главе. Короче говоря, язык XML был создан как раз для пользователей Web, и если правильный документ может быть принят, то есть воспроизведен браузером или

другим агентом, нет необходимости посылать определение типа документа – это просто лишний трафик. Это кажется странным, но тем не менее при некоторых обстоятельствах можно без соответствующего DTD пустить в дело XML-документ, несмотря на то, что агент не знает точно, что означают созданные вами тэги.

Хотя язык XML гораздо проще SGML, фактически он гораздо строже, именно поэтому для XML-файла не обязательно определение типа документа. Так происходит потому, что строгость языка XML позволяет XML-процессору по одному только виду документа типа well-formed сделать заключение о том, какие правила к нему применить. Процессор при этом выстраивает дерево всех вложенных элементов и устанавливает соотношения между всеми его частями. Язык SGML не требует закрывающих тэгов, поэтому невозможно обработать SGML-документ таким же образом, не имея его DTD. Итак, не обязательно тратить пропускную способность канала на пересылку определения типа документа, поскольку правильный XML-документ всегда будет использован или воспроизведен с сохранением функциональных возможностей.

Это не означает, что можно ограничиться беглым просмотром включенного в главу 2 раздела, посвященного определению типа документа. В любом случае желательно, чтобы созданные в XML тэги соответствовали некоторому нормированному DTD. Это нужно для того, чтобы документы, создаваемые и используемые разными людьми, были совместимы друг с другом, то есть подчинялись правилам, установленным в определении типа документа. Соответствие одним и тем же правилам помогает поддерживать структуру сети и вызывает ощущение того, что в рамках одного проекта трудятся сразу много авторов-единомышленников. Определения типа документа также позволяют использовать программу, называемую верифицирующим анализатором (анализатор с проверкой на состоятельность), которая позволяет авторам убедиться в том, что они не нарушают правила DTD. Это будет просто здорово, если подобная программа поможет любому пользователю свести усилия по описанию какого-либо материала до минимума.

Познакомившись с определением типа документа, следует перейти к тому, как оно подсоединяется к XML-документу, и как документ, подсоединенный к таблице стилей, затем форматируется для браузера (рис 1.1). Теперь самое время остановиться на таблицах стилей.

Совет *Не забывайте, что во время подготовки книги к изданию основные браузеры практически не поддерживали язык XML. Несмотря на то, что скорей всего браузер Netscape Communicator 5, а также браузер Microsoft IE5 в какой-то степени начнут все-таки поддерживать XML, существуют способы, с помощью которых можно воспроизводить XML и в нестандартных браузерах. К ним мы вернемся позже, после обсуждения таблиц стилей.*

Таблицы стилей

В отличие от тех методик, идей и предложений, которые обсуждались здесь до сих пор, таблицы стилей ни в коем случае нельзя считать открытием последних

дней. Они стары в той же степени, что и печатное дело. Таблицы стилей – это, по существу, набор правил, устанавливающих, как должен выглядеть документ. Даже во времена ручного набора издатели книг пользовались комплектом письменных инструкций, которые при необходимости подсказывали наборщику, как должна выглядеть страница. Без них рабочий просто не знал, как разметить рукопись. Поскольку наши XML-документы при показе документов в Web-браузере не содержат детальной информации, в каком виде должно выглядеть содержание (при этом обязательно должно быть сохранено разграничение между оформлением и содержанием), нам тоже приходится использовать таблицы стилей, чтобы документы выглядели красиво или были удобны для работы.

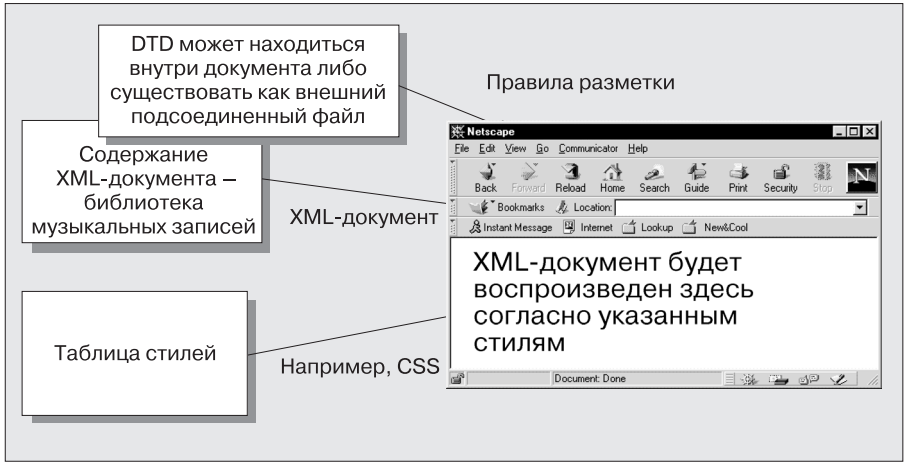


Рис. 1.1. Связь DTD-определений с XML-документом

Преимущества таблиц стилей

Итак, таблицы стилей, являясь необходимым дополнением к DTD и определяя внешний вид документа, имеют ряд преимуществ более общего характера:

- придают документам ясность;
- позволяют уменьшить время загрузки, трафик сети и нагрузку на сервер;
- в случае необходимости могут представить один и тот же источник информации различными способами;
- позволяют изменить вид нескольких файлов изменением только одного файла, который содержит правила о том, как должны выглядеть документы.

Как уже говорилось во введении, с того момента, когда созданием Web-страниц занялись пользователи, не имеющие отношения к науке, чрезмерная озабоченность внешним видом документов привела к созданию огромного числа новых и неудобных стилистических тэгов и атрибутов. В результате размеры файлов стремительно выросли – это явление называют *перегрузкой страниц* (page bloat), – что заметно усложнило чтение кодов и, как следствие, поддержку Web-траниц. Рассмотрим два примера.

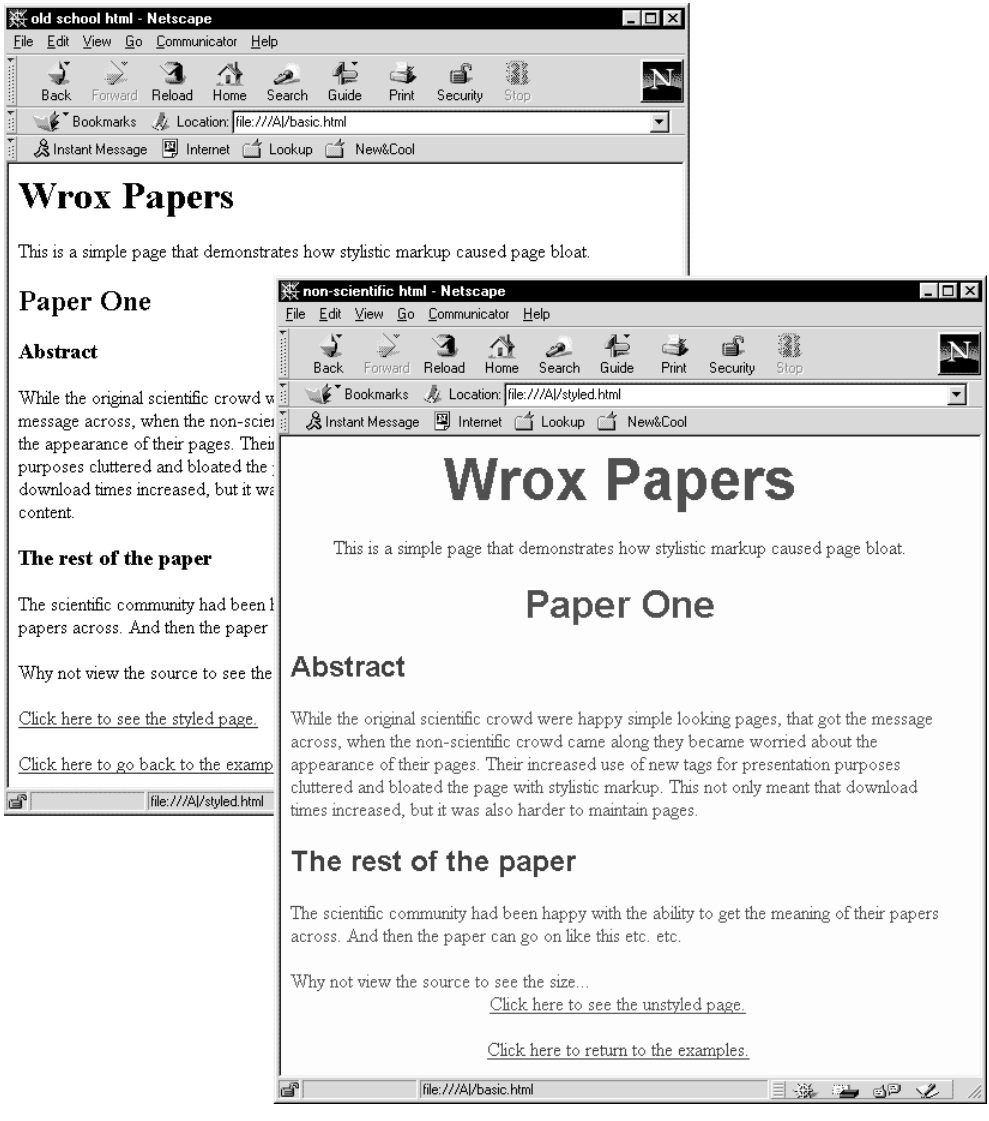


Рис. 1.2. Примеры отображения HTML-документ со стилистической разметкой и без нее

Использование стилистической разметки на второй странице увеличивает размер файла и ухудшает разборчивость кода. Эти страницы можно просмотреть на нашем Web-сайте:

http://webdev.wrox.co.uk/books/1525

Далее показано, как отличаются размеры файлов кода.

```
<HTML>
<HEAD>
<TITLE>old school html</TITLE>
</HEAD>

<BODY>



<H1>Wrox Papers</H1>

This is a simple page that
demonstrates how stylistic
markup caused page bloat.



<H2>Paper One</H2>

<H3>Abstract</H3>


While the original scientific
crowd...
...see their content.

<H3>The rest of the paper</H3>



The scientific community had been
...
```

```
<HTML>
<HEAD>
<TITLE>non-scientific html</TITLE>
</HEAD>

<BODY BGCOLOR="#F5FFFA"
ALINK="#F4A460" LINK="#993200"
VLINK="#556B2F" TEXT="#556B2F">

<H1 ALIGN=CENTER><FONT FACE=ARIAL
SIZE=7>Wrox Papers</FONT></H1>

<DIV ALIGN=CENTER><FONT
COLOR="#993200">This is a simple
page that demonstrates how stylistic
markup caused page
bloat.</FONT></DIV>

<H2 ALIGN=CENTER><FONT FACE=ARIAL
SIZE=6>Paper One</FONT></H2>

<H3><FONT FACE=ARIAL SIZE=5
COLOR="#993200">Abstract</FONT></H3>

While the original scientific
crowd...
...see their content.

<H3><FONT FACE=ARIAL SIZE=5
COLOR="#993200">The rest of the
paper</FONT></H3>

The scientific community had been
...
```

Чтобы поддерживать необходимый внешний облик документов, содержащих более одной страницы, на любом Web-сайте или во внутренней сети (intranet) какой-либо компании или организации необходимо вместе с каждой страницей пересылать браузеру одни и те же правила стилей. Поскольку браузеры помещают данные, полученные из Web-страниц, в кэш, эта многократно повторяемая процедура пересылки одних и тех же правил является пустой тратой времени, ложится тяжким бременем на пропускную способность канала связи, замедляет загрузку документа и затрудняет работу сервера. Таблицы стилей как раз и собирают все правила стилей в один отдельный документ, причем это делается таким образом, чтобы обойтись только одним запросом. Каждая из страниц сайта, обратившись к кэш-памяти браузера, затем может ссылаться на одну и ту же таблицу стилей, которая там хранится. Ясно, что подобная методика существенно сберегает сетевую окружающую среду.

К этому можно добавить, что вместо того, чтобы менять правила стилей на каждой странице, внешний вид всего сайта можно преобразовать изменением лишь одной таблицы стилей, поскольку все стилихранятся в одном файле (рис. 1.3).

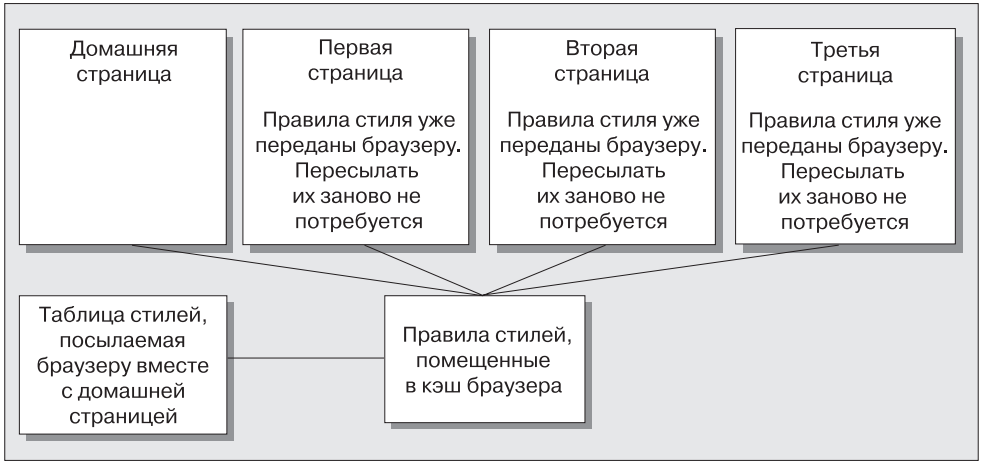


Рис. 1.3. Действие каскадных таблиц стилей

Если вы на нескольких страницах используете одни и те же данные, причем, каждая страница представляет данные различными способами, все что вам надо сделать – это изменить строку, где находятся ссылки на различные таблицы стилей. Примером может служить ситуация, когда в вашем сайте присутствуют различные разделы, в которых одна и та же информация должна быть представлена в особом стиле, соответствующем данной части. Или же, если вам нужно представить информацию для людей с плохим зрением в виде крупного текста, вы можете предложить им ту же самую страницу, но подсоединить к ней стиль с большим размером шрифта.

Каждая из страниц содержит ссылку на таблицу стилей, поэтому агент пользователя знает, откуда ее получить – это похоже на то, как в HTML используются ссылки на рисунок. Существует несколько языков таблиц стилей:

- каскадные таблицы стилей (CSS);
- расширяемый язык таблиц стилей (XSL);
- язык семантики и спецификаций стилей документа (DSSSL);
- XS, также известный как DSSSL-0;

Двумя из них – CSS и XSL – стоит поинтересоваться особо. В свое время DSSSL являлся официальным языком стилей для SGML – это очень мощное и в не меньшей степени сложное средство описания таблицы стилей. По этой причине он и не получил широкого признания среди программистов. Язык XS или DSSSL-0 создавался как упрощенная форма DSSSL, но на практике он все еще оставался слишком сложным, так что в Internet не прижился.

Давайте кратко остановимся на первых двух языках.

Каскадные таблицы стилей

Спецификация каскадных таблиц стилей первого уровня (CSS1) была выпущена консорциумом W3C в конце 1996 года. В некоторой степени этот язык поддерживается как браузером Communicator 4, так и браузером IE4, при этом оба основных производителя браузеров клятвенно обещают полностью поддерживать CSS1 в новых версиях Communicator 5 и IE5.

Совет *Первый уровень спецификации CSS (CSS1) вскоре будет заменен вторым уровнем CSS (CSS2). Рекомендации к выпуску были даны в мае 1998 года. Однако в настоящее время даже первый уровень внедрен еще не полностью, так что придется подождать, пока производители браузеров обеспечат полную поддержку второму уровню CSS.*

Каскадные таблицы стилей уже активно проникают в Web. Такое название появилось вследствие того, что при наличии нескольких таблиц стилей из них обычно формируется каскад, то есть иерархия со свойствами, взятыми из всех таблиц. Любые конфликты разрешаются в соответствии с некоторым набором правил. Каскадные таблицы стилей конструируются достаточно легко, и коль скоро они созданы, не существует предела числу страниц, которые могут их использовать. Чтобы присоединить к документу CSS, достаточно указать в HTML-документе одну лишь строку:

```
<LINK REL="stylesheet" TYPE="text/css" HREF="example.css">
```

В настоящее время консорциум W3C изыскивает способ, с помощью которого XML-документы будут присоединяться к каскадным таблицам стилей. Сообщение по этому поводу находится по адресу:

http://www.w3.org/TR/NOTE-xml-stylesheet

Один из самых распространенных сейчас способов включения CSS выглядит следующим образом:

```
<?xml-stylesheet href="cdlib.css" type="text/css"?>
```

Вот пример каскадной таблицы стилей, который может быть использован совместно с рассмотренной нами библиотекой компакт-дисков:

```
artist {
   display: block;
   font-family: Arial, Helvetica;
   font-weight: bold;
   font-size: 20pt;
   color: #9370db;
   text-align: center;
   }

title {
   display: block;
   font-family: Arial, Helvetica;
   font-size: 20pt;
```

```
    color: #c71585;
    }

format {
    display: block;
    font-family: Arial, Helvetica;
    font-size: 16pt;
    color: #9370db;
    }

description {
    display: block;
    font-family: Arial, Helvetica;
    font-style: italic;
    font-size: 16pt;
    color: #FF1010;
    }
```

При просмотре в браузере наша библиотека компакт-дисков с данной таблицей стилей теперь будет выглядеть так, как показано на рис. 1.4. Изображение было получено с использованием бета-версии браузера IE5, и если вы им пользуетесь, можно попробовать выполнить ту же операцию, используя материал нашего Web-сайта:

http://webdev.wrox.co.uk/books/1525

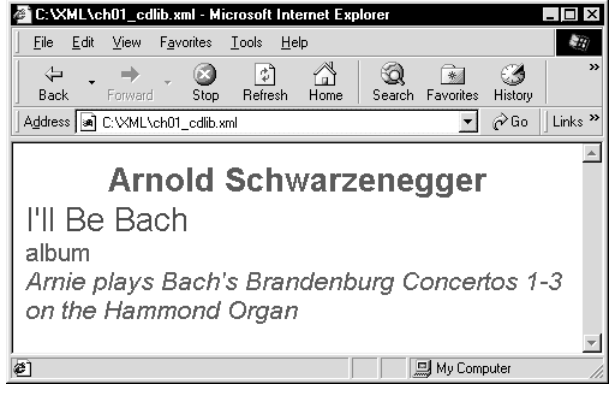


Рис. 1.4. Отображение документа с применением каскадных таблиц стилей

Использование каскадных таблиц стилей дает много преимуществ. Подробнее о таких таблицах стилей можно прочитать в главе 7. Следует иметь в виду, что язык CSS использует фиксированный набор типов разметки, так что здесь вам не удастся создать собственные тэги – CSS не допускает расширения. Для этого существует другое, более мощное средство – язык каскадных таблиц стилей XSL.

Расширяемый язык таблиц стилей XSL

Многие разработчики, пишущие на языке XML, для отображения XML-документов несложной структуры будут вполне удовлетворены функциональными

возможностями CSS, однако нередки случаи, когда пользователи испытывают потребность в более мощных средствах форматирования. Для этого и существует XSL. Этот язык сохраняет те же преимущества, что и CSS, но обладает дополнительными функциональными возможностями.

XSL заимствует приемы как из языка DSSSL, так и из CSS (и DSSSL-0). Он также использует ECMAScript, полученный из JavaScript.

XSL состоит из двух частей:

- словаря XML для определения семантики форматирования;
- языка для преобразования XML-документов.

Язык XSL не только позволяет пользователю определить, как должны выглядеть части документа (то есть установить шрифт, размер, цвет, выравнивание, границы и т.п.), чего собственно можно добиться и с помощью CSS, но, будучи расширяемым, разрешает пользователю создавать свои собственные форматирующие тэги и свойства.

Язык XSL также позволяет авторам в большей степени контролировать представление их документов. Например, с его помощью разрешается добавлять правила, которые указывают порядок расположения разделов. В примере с библиотекой компакт-дисков подобные предписания могли бы определить разный порядок представления данных, то есть можно было бы сделать так, чтобы первым тэгом был `<artist>` или `<title>`. Взаимное расположение самих данных при этом значения не имеет, поскольку тэги могут быть интерпретированы, а порядок их следования изменен.

Во время подготовки сборника язык XSL еще находился в консорциуме W3C на стадии рабочего проектирования. Первоначально это было совместное предложение компаний Microsoft, Inso Corporation и ArborText, однако рабочий проект 1.0 превосходит этот замысел. Наш совет – постоянно следите за развитием XSL. За информацией по этому вопросу следует обращаться к XSL-странице сайта консорциума W3C:

http://www.w3.org/Style/XSL/

Каскадные таблицы стилей и язык XSL

Каскадные таблицы стилей, как, впрочем, и XSL, скоро появятся в Web. Места им хватит, так как они преследуют разные цели (см. табл. 1.1). Язык XSL позволяет автору управлять сложным форматированием, при котором содержание документа может появляться в нескольких местах. Скажем, в примере с библиотекой компакт-дисков имя исполнителя записи может также появляться в динамически генерируемой таблице, демонстрирующей каталог исполнителей, или использоваться в качестве заголовка страницы. В то же время каскадные таблицы стилей применяются для динамического форматирования текущих документов, предназначенных для многих средств передачи – не только для визуального воспроизведения браузерами, но и звукового воспроизведения, воспроизведения системами чтения для слепых и т.п.

Таблица 1.1. Сравнение каскадных таблиц стилей и расширяемого языка таблиц стилей

	CSS	XSL
Может быть использован вместе с HTML	Да	Нет
Может быть использован вместе с XML	Да	Да
Является языком преобразования	Нет	Да
Синтаксис	CSS	XML

Совет *Более подробную информацию о таблицах стилей вы можете найти в книге Professional Style Sheets for HTML and XML («Профессиональные таблицы стилей для HTML и XML»), изданной Wrox Press (ISBN 1-861001-65-7).*

Анализаторы

Авторам не хотелось бы затевать на этих страницах философские споры. Действительно, когда наш XML-файл сохраняется в текстовом формате, который может прочитать человек, нелепо полагать, тем более употреблять выражение, что компьютеры «способны читать» файл. Они всего лишь интерпретируют его, однако и в дальнейшем мы будем вынуждены прибегать к терминологии, связанной скорее с деятельностью человека, нежели машины. Это касается и анализатора, то есть программы, которая должна помочь компьютеру интерпретировать XML-файл, представляя его в формате, воспринимаемом компьютером. В свою очередь, XML-приложения помогают как нам, так и анализаторам интерпретировать файл с помощью содержащейся в них информации о природе фрагментов текста, заключенных в тэги.

Спецификация XML определяет два компонента, необходимые для практического использования XML: *XML-процессор* и *приложение*. *Анализаторы* (parsers) играют роль XML-процессора, они загружают XML-файл и другие нужные файлы, проверяют, является ли XML-документ правильным или состоятельным (в зависимости от типа анализатора) и строят структуру дерева документа, которая может быть передана приложению. Именно эта структура дерева может затем быть практически использована компьютером. Приложение представляет собой ту часть программы, которая обрабатывает данные, находящиеся в дереве документа, созданном процессором.

На практике анализатором в большинстве случаев является обычный компонент, который программисты вызывают при построении приложения. Браузер IE5 как таковой включает в себя XML-анализатор, называемый MSXML.

Совет *Помните, мы говорили, что XML-документам не всегда нужно определение типа документа? Именно анализаторы способны формировать дерево документа без определения типа документа, что позволяет XML-файлам быть функциональными самим по себе.*

HTML-браузеры соединяют в себе и процессор, и приложение. Поскольку язык HTML представляет собой набор требований, управляющий тем, как должен быть воспроизведен документ, существуют строгие правила интерпретации файла. Однако если DTD и таблица стилей отделены от XML-файла, приложения могут использовать один и тот же документ в различных целях. Таким образом, тип приложения не ограничен одними лишь браузерами; приложение может быть и текстовым редактором, и драйвером специального принтера, печатающего документы для слепых, и многим другим. Приложение не обязательно должно напрямую использоваться человеком, оно может являться автоматическим средством, например системой, которая посылает письмо, когда происходит определенное событие – допустим, превышение вами кредита в банке. Оно может даже добавлять плату за это письмо к вашему счету.

В то время как все описанное выше является основным назначением анализаторов, они выполняют еще одну очень полезную функцию. Мы уже видели, что поскольку XML является гораздо более строгим языком, чем SGML, анализаторы можно использовать как средство, помогающее авторам, пишущим на XML, проверять свои документы, а программному обеспечению, редактирующему XML-файлы – гарантировать, что оно создает документы, соответствующие правилам. Может показаться, будто создание XML-документов предъявляет гораздо больше требований к авторам, чем создание HTML-документов. Однако преимущества XML в возможностях обработки и гибкости преобладают над недостатками, связанными с его строгостью – обязательными закрывающими тэгами и необходимостью тщательного соблюдения синтаксиса. Когда вы написали длинный и сложный XML-документ и обнаружили, что он не работает, задача просмотра кода и поиска ошибки может показаться устрашающей. В таком случае, прежде чем использовать XML-документ, имеет смысл пропустить его через анализатор, который проверит документ вместо вас и укажет ошибки – такие как, например, отсутствие закрывающего тэга.

Совет *Все меньше и меньше людей пишут HTML-документы вручную, с одним лишь Notepad. В основном используются программы, автоматизирующие труд автора. Того же самого можно ожидать и в случае с XML. В ближайшем будущем должны появиться редакторы XML.*

Существует два основных типа анализаторов. Относительно простые *неверифицирующие анализаторы* (non-validating parsers) только тестируют, является ли документ правильным, и имеют размер всего 30–40 Кб, в то время как более сложные *верифицирующие анализаторы* (validating parsers), кроме того, проверяют состоятельность документа, используя DTD.

Существует несколько типов анализаторов, которые можно бесплатно получить через Internet, и мы настоятельно рекомендуем вам раздобыть хотя бы один. Если у вас есть возможность, загрузите несколько анализаторов, поскольку различные анализаторы по-разному дают сообщения об ошибках. Рассмотрим некоторые доступные анализаторы.

Совет *Есть дополнительное преимущество проверки точного соответствия документа требованиям спецификации. Оно состоит в том, что отпадает необходимость вводить в агента обширный код только для того, чтобы он мог справиться с плохо написанными документами. Как известно, что текущие версии основных Web-браузеров включают код, который позволяет им просматривать HTML-файлы, не находящиеся в точном соответствии со спецификацией HTML.*

Анализатор Lark

Анализатор Lark, написанный Тимом Бреем (Tim Bray, один из редакторов спецификации XML), был одним из лучших первых средств для проверки документов. Он представляет собой неверифицирующий XML-процессор, написанный на языке Java, для использования которого необходима *виртуальная машина Java* (Java Virtual Machine).

Совет *Java Virtual Machine включена в пакет Microsoft Visual J++ for Windows, в пакет Macintosh Runtime for Java SDK, в OS/2 Warp версии 4; еще можно загрузить Sun Java Development Kit (JDK) или, как упрощенный вариант, – Java Runtime Environment (JRE) по адресу http://java.sun.com/.*

Несмотря на то, что анализатор Lark не имеет визуального интерфейса с пользователем, он компактен и надежно работает. Этот анализатор эффективно строит дерево документа и сопоставляет тэги, однако не проверяет, является ли документ состоятельным, путем сопоставления документа с соответствующим DTD. Тим также написал верифицирующий XML-анализатор, основанный на том же коде, что и Lark, и называемый Larval. Вы можете бесплатно загрузить копии обеих программ на сайте

http://www.textuality.com/Lark/

Анализатор XP

Анализатор XP Джеймса Кларка (James Clark) представляет собой неверифицирующий анализатор языка XML 1.0, написанный на языке Java. Он проверяет только, является ли документ правильным. Анализатор XP можно загрузить по адресу

ftp://ftp.jclark.com/pub/xml/xp.zip

Анализатор XML компании Microsoft

Анализатор XML, созданный Microsoft и написанный на языке Java, представляет собой механизм, тестирующий, является ли документ правильным и по выбору предоставляет возможность определить состоятельность документов. Обработанный анализатором XML-документ оформляется в виде дерева с помощью ряда методов Java, над стандартизацией которых Microsoft работает совместно с консорциумом W3C. Если вы используете этот анализатор, имеет смысл следить за

его развитием, поскольку Microsoft постоянно совершенствует свои разработки. Чтобы получить более подробную информацию и загрузить версию анализатора, посетите сайт

http://www.microsoft.com/xml/

Конечно же, существует множество анализаторов, доступных через Internet, но мы не можем рассмотреть их все. Почему бы вам не запустить вашу любимую поисковую машину, чтобы найти другие анализаторы?

Создание ссылок в XML

По ходу изучения этой книги читателям придется освоить очень много нового материала, поэтому вам будет приятно узнать, что способ создания ссылок, который вы изучили в HTML, работает и в XML. Вы все еще можете организовывать ссылки следующим образом:

```
<a href= "http://webdev.wrox.co.uk">Wrox Web Developer</a>
```

Конечно, поскольку мы используем XML, элемент `<a>` должен быть описан в определении типа документа так же, как и атрибут `href`, несмотря на то, что `href` и `HREF` представляют собой ключевые слова, зарезервированные в спецификации создания ссылок. Однако поскольку мы пишем на XML, можно использовать более информативный тэг, например:

```
<webdevlink href="http://webdev.wrox.co.uk">Wrox Web Developer</webdevlink>
```

Эта конструкция должна быть описана в определении типа документа следующим образом:

```
<!ELEMENT webdevlink (#PCDATA)>
<!ATTLIST webdevlink
   xml:link CDATA #FIXED "simple"
   href CDATA #REQUIRED
>
```

Сочетание `xml:link` является зарезервированным ключевым словом языка XML, используемым как атрибут для определения ссылок; он может принимать значения `"simple"` (простой) или `"extended"` (расширенный). Так как значение атрибута `xml:link` описано как `#FIXED` (закрепленный), атрибут `xml:link` *не обязательно* включать в каждый экземпляр элемента `webdevlink`. Он подразумевается в каждом тэге `<webdevlink>`, поэтому мы можем указать:

```
<webdevlink href="http://webdev.wrox.co.uk">Wrox Web Developer</webdevlink>
```

вместо того, чтобы писать:

```
<webdevlink xml:link="simple" href="http://webdev.wrox.co.uk">Wrox Web
Developer</webdevlink>
```

Предложения по созданию ссылок в XML достигли статуса рабочего проекта в марте 1998 года, и как вы уже могли заметить, ссылки записываются на языке XML. Первоначально язык организации ссылок был известен как XLink, затем как

XML-Link, а текущим рабочим названием для языков создания ссылок, находящихся в процессе разработки, является термин *расширяемый язык создания ссылок* (Extensible Linking Language, XLL). В основу XLL легли языки SGML, HyTime и Text Encoding Initiative (TEI) – два последних применяются в качестве методов организации ссылок в SGML.

Если вы довольствуетесь простыми ссылками, то есть теми, которые используются в HTML, вам не придется обременять себя лишними знаниями по этому предмету. Однако следует учесть, что возможности создания ссылок в XML гораздо более развиты, чем в HTML. Вы найдете множество применений новым типам ссылок и скорее всего захотите узнать о них побольше, а лучше – всю полезную информацию. Используя XLL, вы сможете:

- определять ваши собственные ссылочные элементы;
- использовать любой элемент в качестве ссылочного элемента;
- создавать двунаправленные ссылки с соотношениями типа «один-ко-многим» и «многие-к-одному»;
- определять способ обхода – то, в каком порядке пользователи будут продвигаться по ссылкам;
- создавать базы ссылок для определения и управления ссылками вне документов, к которым они относятся;
- группировать ссылки;
- делать ссылку, адресатом которой является часть документа, откуда исходит эта ссылка (трансклюзия).

Существует два основных типа ссылок, различаемых по способу хранения и вызова:

- *встроенные ссылки* (inline links), то есть ссылки, которые определяются в том же месте, где они инициируются;
- *внешние ссылки* (out-of-line links), то есть те, которые хранятся во вспомогательном файле – базе данных ссылок.

Организация ссылок не включена в спецификацию XML. Напротив, существует две отдельные спецификации создания ссылок в XML:

- *X-ссылки* (XLinks) для организации ссылок из отдельно взятого XML-документа на другие XML-документы;
- *X-указатели* (XPointers) для адресации внутренней структуры XML-документов, обеспечивающей ссылки на элементы, строки символов и другие части XML-документов, включая также части, не имеющие явного атрибута ID.

Любой элемент может являться X-ссылкой. Элементы, включающие в себя ссылки, называют *ссылочными элементами* (linking elements)*ссылки*. Поскольку спецификации создания ссылок написаны на XML, вы можете разрабатывать ваши собственные тэги ссылок, которые описывают ссылку так, как показано в примере с тэгом `webdevlink`. Тогда ссылка будет описывать соотношения между объектами или частями объектов данных.

Вы можете найти спецификацию XLinks на сайте консорциума W3C:

http://www.w3.org/TR/1998/WD-xlink-19980303

X-указатели используются в сочетании с унифицированными идентификаторами ресурсов (URI) для определения части документа (или подресурса). Они могут быть использованы для создания ссылки на определенную часть документа как целого или для организации ссылки, выбирающей только часть документа (в противопоставление целому документу). Любая ссылка, которая адресует часть документа, должна быть в форме X-указателя. Однако явное включение атрибутов ID в язык X-указателей тем же способом, каким это делается в HTML, не обязательно. В то же время эти атрибуты предоставляют возможность создания особых ссылок на отдельные части XML-документов, таких как элементы, строки символов и другие.

Вы можете найти спецификацию X-указателей на Web-странице:

http://www.w3.org/TR/1998/WD-xptr-19980303

При использовании языка HTML, если вы хотите создать ссылку на определенную часть документа, приходится изменять сам документ – в него должны быть включены опорные метки. Эта процедура не является необходимой при использовании языка XLL и X-указателей. Язык XLL также позволяет приложениям пользователя устанавливать связи между документами и частями документов в отсутствие самого пользователя.

Комментарии в XML

Комментарии в XML такие же, как и в HTML. Они помещаются внутри тэгов следующего вида:

```
<!--комментарий-->
```

Грамотное использование комментариев является своего рода искусством. Способность применять их развивается по мере профессионального роста программиста. Комментарии бывают необходимы, когда вы возвращаетесь к документу через пару месяцев и те записи, которые раньше казались очевидными, теперь уже далеко не так понятны. Грамотное употребление комментариев означает также, что и другие люди смогут с легкостью пользоваться вашим документом. Это особенно важно при создании определений типа документа, поскольку вы можете пожелать, чтобы вашими определениями пользовались и другие. О правильном размещении комментариев в XML рассказывается в главе 2.

Новые птенцы в нашем гнезде

К настоящему моменту вы, должно быть, догадались, что XML и другие относящиеся к нему спецификации далеки от завершения. В этом разделе мы познакомим вас с некоторыми новыми спецификациями, которые сейчас являются предметом обсуждения в XML-сообществе. Это никак не означает, что какая-то из тех составляющих, которые предполагается улучшить с помощью новых спецификаций, внезапно исчезнет. Когда у вас возникнет ясное представление о том, как

строятся XML-приложения, вам будет нетрудно следить за развитием этих новых спецификаций. В этом разделе мы рассмотрим предложения, относящиеся к:

- пространствам имен XML (XML Namespaces);
- XML-данным (XML-Data);
- описанию содержания документа (Document Content Description).

Пространства имен XML

Одним из преимуществ XML, которое мы упоминали во введении, является возможность создания составных документов из нескольких отдельных источников. Скорее всего, чтобы освободить нас от лишней работы, для этого будут использованы модули программного обеспечения. Существуют, однако, две проблемы, которые эти программные модули должны будут преодолеть, чтобы создавать составные документы:

- распознавание разметки, которую им необходимо обработать (тэгов и атрибутов);
- преодоление конфликтов имен в разметке.

Первая проблема ясна сама по себе: программа обязана знать, к какой части документа она обращается. Вторая проблема менее очевидна. В то время как возможность использования документов из многих источников кажется очень привлекательной, существует риск, что при создании каждым автором своих собственных тэгов возникнут *конфликты имен* (name collisions), используемых для тэгов и атрибутов. Если одни беззаботно создают свои собственные тэги, а другие следуют стандарту, очень скоро возникает ситуация, когда в двух файлах одно и то же имя тэга будет использовано для описания различных предметов. В нашем примере с библиотекой компакт-дисков мы использовали элемент `<format>` для описания того, является ли компакт альбомом или одиночным диском. Возможно, что в другом похожем документе тэг `<format>` будет описывать, на чем сделана запись: на компакт-диске, грампластинке, кассете, DVD, MiniDisc'е и т.п. Предложение по *пространствам имен XML* (XML Namespaces) направлено на решение этих двух проблем.

Для преодоления этих препятствий синтаксические конструкции документа должны иметь глобально уникальные имена. Один из способов, с помощью которого можно это сделать, состоит в определении уникального пространства имен для типов и атрибутов элементов.

Ниже приведено определение пространства имен в XML, данное консорциумом W3C.

Цитата

Пространство имен в XML представляет собой группу имен, идентифицируемую унифицированным указателем ресурса (URI) и используемую в XML-документах для типов и атрибутов элементов. Пространство имен в XML отличается от пространств имен, обычно использующихся в компьютерных дисциплинах, тем, что оно имеет внутреннюю структуру и, говоря математическим языком, не является множеством.

Пространства имен ставят в соответствие унифицированному указателю ресурса (URI) префикс, используя следующий синтаксис:

```
xmlns:[prefix]= "[URI of namespace]"
```

Таким образом, мы можем записать:

```
xmlns:wroxcds="http://webdev.wrox.co.uk/books/1525"
```

чтобы объявить префикс `wroxcds`, причем тэги элементов из указанной области Internet внутри составного документа будут определены единственным образом.

Может показаться, что все это означает большую дополнительную работу. Не отчаивайтесь, не так все плохо. Не вдаваясь пока в детали, отметим следующее: если где-то выше в данном документе определено пространство имен, элемент `<format>` не становится намного сложнее и в конечном счете будет выглядеть так:

```
<wroxcds:format>album</wroxcds:format>
```

Как вы уже догадались, предложение по пространствам имен еще может измениться в процессе его обсуждения консорциумом W3C.

Рабочий проект по пространствам имен XML вышел в свет в августе 1998 года. Мы вернемся к вопросу о пространствах имен в главе 4 и, кроме того, следите за изменениями на странице

http://www.w3.org/TR/1998/WD-xml-names

XML-схемы

Схемы (Schemas) определяют характеристики классов объектов. Таким образом, в XML схема просто является указанием способа разметки документа. Определение типа документа представляет собой хороший пример схемы в том отношении, что оно описывает класс документов, а также типы элементов и атрибутов в этом классе.

Во время написания книги следующие предложения по XML-схемам в консорциуме W3C были все еще на стадии сообщений, несмотря на то, что вызвали немалый интерес в XML-сообществе:

- *XML-данные* (XML-Data);
- *описание содержания документа* (Document Content Description, DCD) для XML.

Консорциум W3C сформировал рабочую группу, в обязанности которой входят просмотр и рецензирование различных предложений по XML-схемам. Однако сейчас еще слишком рано говорить о том, что будет решено по поводу этих предложений. Первым шагом является создание технического задания, на основе которого будет проходить последующее обсуждение. Таким образом, не стоит ожидать, что XML-схемы сохранятся в их настоящем виде. Однако мы рассмотрим их в нашей книге, чтобы познакомить вас с концепцией схем, написанных с использованием синтаксиса XML, который, как вы увидите, имеет некоторые преимущества по сравнению с традиционными определениями типа документа.

XML-схемы появились благодаря уязвимым местам традиционного DTD, которое мы видели ранее (и к которому мы вернемся в последующих главах). Традиционное DTD критикуется в отношении следующих характеристик:

- оно использует синтаксис, отличный от синтаксиса XML;
- хорошее DTD трудно создать;
- оно имеет ограниченную способность к описанию;
- оно не расширяемо;
- оно не описывает XML как источник данных;
- оно не обеспечивает поддержку предложений по пространствам имен.

Мы вернемся к обсуждению этого предмета в главе 3. В данный момент давайте кратко остановимся на том, что такое XML-данные и описание содержания документа (DCD).

XML-данные

Предложение по *XML-данным* было представлено в консорциум W3C 11 декабря 1997 года как способ описания схем с использованием синтаксиса XML. Как вы увидите в следующей главе, в определении типа документа используется модифицированная форма записи, называемая *расширенной формой Бэкуса-Наура* (Extended Backus-Naur Form, EBNF). Простота *не является* сильной стороной этой формы. Поэтому идея описания схем в XML, как это делается в XML-данных, была принята благосклонно.

Когда мы имеем дело с относительно простыми документами, DTD может показаться подходящим способом представления схем. Однако в Web существует потребность рассматривать XML не только как способ разметки документов, но и как данные. При этом, благодаря возможности ставить в соответствие нашим данным различные схемы, можно получить значительный выигрыш в эффективности использования сети. Информацию по XML-данным смотрите на сайте консорциума W3C:

http://www.w3.org/TR/1998/xml-data

или на сайте Microsoft:

http://www.microsoft.com/xml

Описание содержания документа для XML

Эта схема для XML-документов еще «зеленее». Она была представлена в консорциум W3C только в июле 1998 года. Описание содержания документа (DCD) также использует синтаксис XML. Основы синтаксиса приведены в главе 3 вместе со ссылками на источники, в которых можно найти дополнительную информацию по XML-схемам. В синтаксисе DCD используется модифицированный синтаксис формата описания ресурсов (Resource Description Framework, RDF). Описание содержания документа включает подмножество проекта XML-данных в виде, совместимом с RDF.

Вы можете найти сообщение по схеме DCD на Web-странице

http://www.w3.org/TR/NOTE-dcd

Изучение синтаксиса XML-данных и DCD даст вам основу для понимания других XML-схем, использующих синтаксис XML. Если у вас появится желание следить за их развитием, вы можете обратиться к указанному выше Web-сайту.

Просмотр XML-файлов

В быстро расширяющемся пространстве XML есть область, в которой новички очень скоро теряют путеводную нить и начинают плутать. Поэтому есть смысл подробнее осветить проблему представления XML-файлов в Web. При знакомстве с таблицами стилей мы рассмотрели только часть этого материала. Теперь вы немного знакомы с тем, каким образом правила стилей применяются к XML-файлу. В этом разделе мы остановимся на просмотре XML-файла в браузере.

Просмотр HTML-файлов в Web относится к тем процессам, к которым все привыкли, и поэтому он кажется очень простым. Все, что надо сделать – это ввести URL и нажать **Enter** или подвести курсор к ссылке, щелкнуть кнопкой мыши – и увидеть новую страницу. Чтобы разобраться с механизмом вывода XML-файлов, давайте кратко остановимся на том, как это делается в HTML.

Вывод HTML-файлов

Ниже приведено упрощенное, но достаточное для наших целей описание того, как браузер получает и показывает страницу:

- HTML-документ, запрошенный клиентом, передается ему с помощью протокола HTTP (или средствами файловой системы, если документ запрашивается не через Internet);
- браузер освобождает документ от тэгов и создает массив содержаний элементов;
- браузер должен распознать, что означает каждый тэг, найти соответствующий тэгу стиль – вспомните, что браузер уже знает HTML (а именно, определение типа документа HTML);
- затем браузер выводит содержание на экран.

XML-браузеры

К сожалению, в XML все не так просто. Во время написания книги продукты основных производителей браузеров (вплоть до четвертой версии) не позволяли непосредственно просматривать XML тем же способом, каким можно просматривать HTML. Нельзя просто открыть браузером XML-файл и просмотреть его содержимое. Netscape и Microsoft обещали поддержку XML в своих браузерах Communicator 5 и IE5, однако уровень поддержки все еще не совсем ясен – в последующих главах мы обсудим это подробнее. (Бета-версии браузера IE5 обеспечивают существенную поддержку основной спецификации XML 1.0, объектной модели документа (Document Object Model), пространств имен (Namespaces), каскадных таблиц стилей (CSS) и расширяемого языка таблиц стилей (XSL)). Тем не менее, не отчаивайтесь. Это еще не означает, что ваши файлы с расширениями `xml` и `xsl/css` невозможно просматривать в браузерах Communicator 4 и IE4. Это означает только, что для просмотра их нужно будет преобразовать в HTML – или статически (перед просмотром) или динамически (по мере обращения к ним). Но

если бы браузеры могли демонстрировать XML непосредственно, это происходило бы аналогично просмотру HTML:

- браузер должен был бы получить XML-файл, освободить его от тэгов, и создать массив содержаний элементов;
- поскольку тэги в XML не предопределены, браузер должен был бы затем найти информацию о стиле каждого элемента в таблице стилей;
- для воспроизведения документа HTML-браузер имеет специальное окно. Однако XML-браузер, возможно, будет не только сам воспроизводить документ, но и преобразовывать его в различные форматы для демонстрации в других приложениях, например, в формат RTF для просмотра в текстовом редакторе.

Поскольку XML-браузер не имеет никакой встроенной информации о том, как должен быть воспроизведен XML-документ, процесс применения стилей, как мы уже видели, абсолютно необходим. Теперь остановимся на том, как XML-документ может быть преобразован в HTML для просмотра в обычном Web-браузере.

Статическое преобразование

Статическое преобразование происходит до того, как файл попадает на Web-сервер. Оно включает использование программы, которая соединяет XML-файл с таблицей стилей и создает HTML-файл, который можно просмотреть в браузере. Именно HTML-файл затем и помещается на сервер для обслуживания запросов.

Одним из примеров такого типа программ является Microsoft MSXSL Command Line Utility, которая соединяет XML- и XSL-файлы и создает HTML-файл. К сожалению, она работает только под Windows 95/98 и Windows NT (только на процессорах x86), а кроме того ей необходим браузер IE4. Ее можно найти по адресу:

http://www.microsoft.com/xml/xsl/downloads/msxsl.asp

Вот, вкратце, что делает эта программа. В командной строке надо набрать:

```
C:\xslproc\msxsl -i xmlfile.xml -s xslfile.xsl -o result.htm
```

Тем, кто пребывает в счастливом неведении и не представляет, что такое командная строка, следует открыть в Windows 9x меню **Start** ⇒ **Run** (Пуск ⇒ Выполнить) и набрать точно такую же строчку, подставляя правильные имена файлов и папок, в которых находятся файлы. Чтобы команда была выполнена, файлы должны находиться в той же папке, что и программа.

Вместо `-i` надо подставить имя входного XML-файла, вместо `-s` – имя файла таблицы стилей XSL, вместо `-o` – имя выходного HTML-файла. Выходной HTML-файл можно затем поместить на Web-сервер или просмотреть HTML-браузером. Правила стилей будут применены.

Менее распространенный способ состоит в запуске специальной программы, загрузки в нее вашего XML-файла и последующего применения к нему правил стилей вручную. Примером такой программы является Cormorant XML Parser (рис. 1.5), написанный одним из авторов этой книги – Фрэнком «Бумером» умфреем. Эта программа находится в бесплатном доступе на нашем Web-сайте:

http://webdev.wrox.co.uk/books/1525

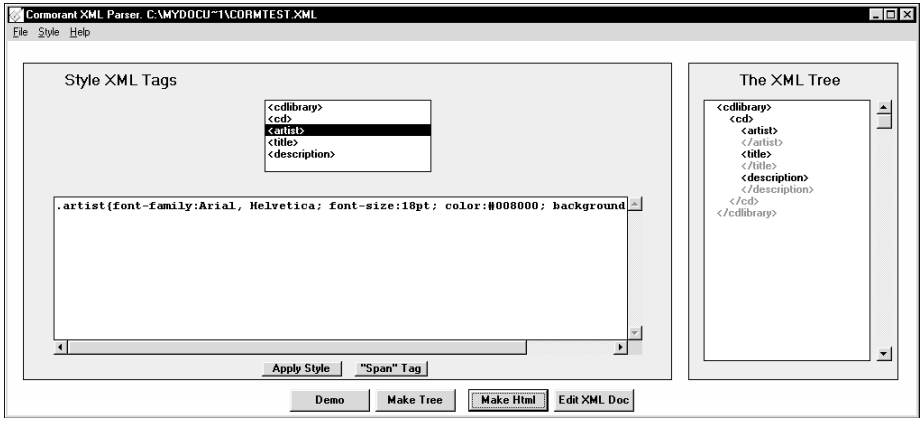


Рис. 1.5. Окно программы Cormorant XML Parser

После того как вы открыли XML-файл (показанный в правой части окна – **The XML Tree** (Дерево XML)), можно применять правила стилей к каждому тэгу в левой части окна, указывая щелчком мыши соответствующий тэг и вводя правила стиля в пустые скобки. Затем, чтобы внести все сделанные изменения, следует щелкнуть по кнопке **Apply Style** (Применить стиль). В завершение щелкните по кнопке **Make HTML** (Создать HTML), которая откроет Notepad (Блокнот) с готовым HTML-файлом.

Если сохранить его с расширением `html`, можно просмотреть первоначальный XML-файл в HTML-браузере (рис. 1.6).

На рис. 1.7 мы видим, как отображается HTML-документ.

Хотя статическое преобразование и неплохой способ для подготовки статических XML-документов, оно не слишком подходит, если вы хотите использовать все возможности XML. Создание динамических страниц, которые используют данные ваших XML-файлов, предоставляя различные файлы разным пользователям, требует динамического метода преобразования.

Динамическое преобразование

Динамическое преобразование XML-файлов позволяет предоставлять разным пользователям различные страницы. Существует несколько способов динамического преобразования, которые используют следующие программы для преобразования файлов с расширениями `xml` и `css/xsl` в HTML по мере необходимости:

- MSXSL Command Line Utility;
- ActiveX Control (элемент управления ActiveX Control);
- Java Applet (апплет Java);
- программа JavaScript.

Повторяем, все эти программы находятся в бесплатном дотупе. Вместо того чтобы рассказывать обо всех программах, мы пкажем, как работает одна из них – MSXSL ActiveX Control.

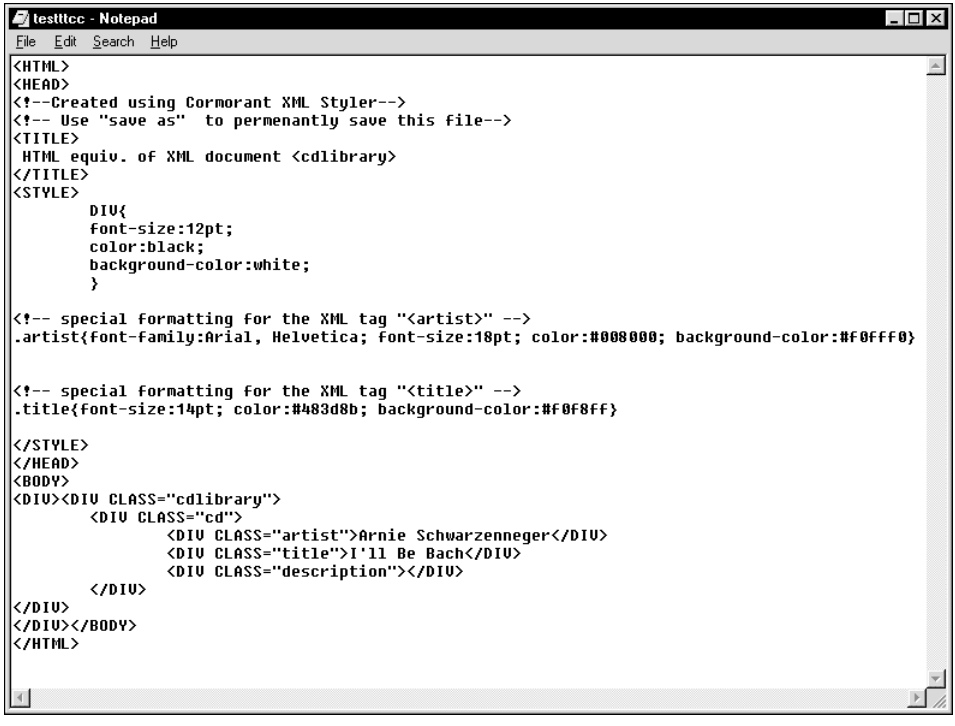


Рис. 1.6. Текст HTML, разработанный на Cormorant XML Parser

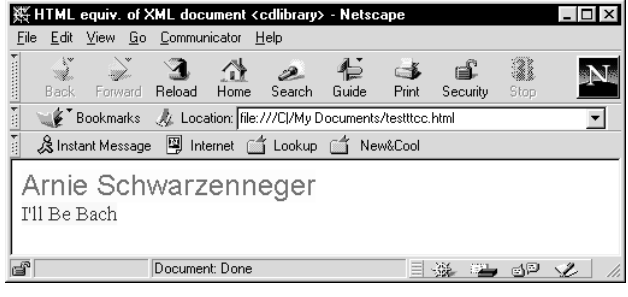


Рис. 1.7. Отображение текста HTML в браузере

Динамическое преобразование с помощью MSXSL ActiveX Control

Элемент управления MSXSL ActiveX Control основан на том же коде, что и сервисная программа командной строки MSXSL, с которой мы уже познакомились. Повторяем, у обеих программ общий недостаток: они могут быть использованы только в среде Windows 9x/NT, и работают только с браузером IE4.x. Однако преимущество MSXSL ActiveX Control состоит в том, что для работы с ним вам не нужно много знать о Java или JavaScript.

Элемент управления ActiveX Control может быть встроен в основную HTML-страницу. В этом случае ActiveX Control загружает XML-файл вместе с его таблицей стилей и преобразует их в HTML. Несложный скрипт JavaScript показывает результат обработки на HTML-странице. Таким образом, HTML-страница является всего лишь местом расположения элемента управления ActiveX Control и оболочкой для возвращаемого кода. Тот же самый ActiveX Control может быть использован на Web-сервере.

Вы найдете MSXSL ActiveX Control по сайте

http://www.microsoft.com/workshop/c-frame.htm#/xml/default.asp

Рассмотренная возможность динамического преобразования является наиболее простым механизмом. Если требуется обеспечить совместимость с различными браузерами, следует рассмотреть два других способа (JavaScript и Java Applets) или исследовать возможность преобразования в HTML на стороне сервера. Последнюю возможность мы рассмотрим в главе 9.

XML в реальном мире

До сих пор мы говорили о XML в общих словах, а теперь рассмотрим реальные примеры использования XML.

Формат определения канала

С того момента, когда стали появляться технологии принудительной доставки информации (push-технологии), производителям браузеров понадобился способ, с помощью которого можно было описывать содержание проталкиваемой информации. Формат определения канала (Channel Definition Format, CDF) представляет собой основанный на XML язык разметки, который позволяет авторам Web-сайтов давать информацию подписчикам об изменениях, в той или иной степени происшедших на сайте. Формат определения канала был включен в браузер IE4 и значительно способствовал росту популярности XML. Документы, посылаемые в формате CDF, соответствуют определению типа документа CDF.

CDF-файлы подсоединяются к HTML- или ASP-файлам сайта. При этом CDF-файлы находятся отдельно от указанных файлов. Таким образом, нет необходимости переписывать сайт, чтобы добавить в него CDF.

Заметим, что формат определения канала отличается от метода, который использует браузер фирмы Netscape. На рис. 1.8 в окне браузера IE4 вы видите наш канал WebDev. Мы посвятим каналам главу 13.

Химический язык разметки

Химический язык разметки (Chemical Markup Language, CML) создан Питером Меррей-Растом (Peter Murray-Rust) в качестве технической поддержки работы с информацией о молекулах. Из-за сложности предмета весь массив подобной информации не мог быть адекватно воспроизведен с помощью обычного языка HTML. Хотя первоначально химческий язык разметки был SGML-приложением, впоследствии он перешел к использованию XML в качестве стандарта развития.

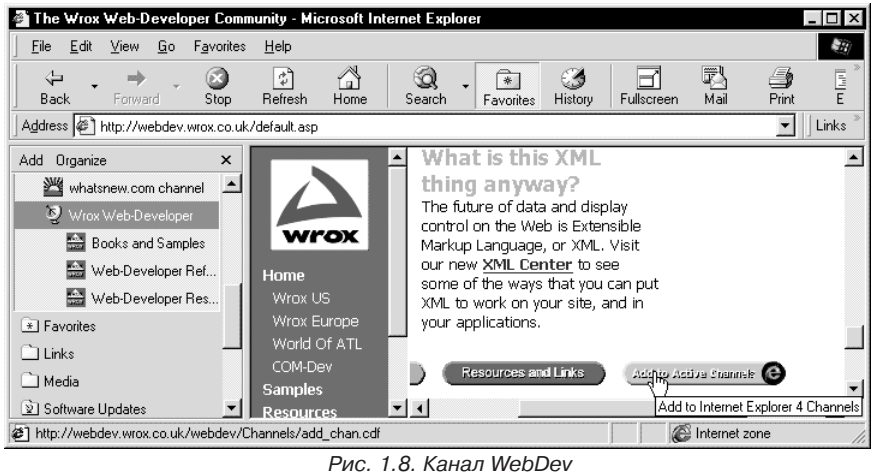


Рис. 1.8. Канал WebDev

Язык CML пока является наиболее приближенным к состоянию промышленного стандарта в XML-сообществе. Он позволяет исследователям хранить химические данные в форме, которая может быть использована повторно. Поскольку CML содержит характерные для химии термины, такие как молекулы, атомы, связи, кристаллы, формулы, последовательности, симметрии, реакции и т.п., использование такого XML-приложения очень удобно для химиков. Способ, с помощью которого он обрабатывает объекты, позволяет легко искать и индексировать документы с помощью компьютера. Поскольку CML написан на XML, он не зависит от платформы, в отличие от наиболее распространенных двоичных форматов, используемых в химии.

Профессор Меррей-Раст также создал JUMBO – первый универсальный XML-браузер, написанный на Java. (Заметим, однако, что он не подходит для воспроизведения XML-страниц в таком же виде, в каком это делают более популярные Web-браузеры). На рис. 1.9 показано окно браузера JUMBO, демонстрирующее документ, написанный на CML.

Браузер JUMBO можно найти по адресу:

http://www.vsms.nottingham.ac.uk/vsms/java/jumbo/

Открытый финансовый обмен

Язык открытого финансового обмена (Open Financial eXchange, OFX) является результатом сотрудничества компаний CheckFree, Intuit и Microsoft в разработке языка, позволяющего безопасно производить финансовые транзакции в Web с помощью определения типа документа для OFX. Как и химический язык разметки, OFX первоначально был SGML-приложением, но в настоящее время использует синтаксис XML. Предложение по OFX включает также вопросы безопасности при пересылке номера кредитной карты по Internet. В нем обеспечивается поддержка механизма *уовня безопасных подключений* (Secure Sockets Layer, SSL), а также способов шифрования с открытым и закрытым ключом, которые лежат в основе SSL. Язык OFX организован таким образом, что может быть встроен в большие приложения.

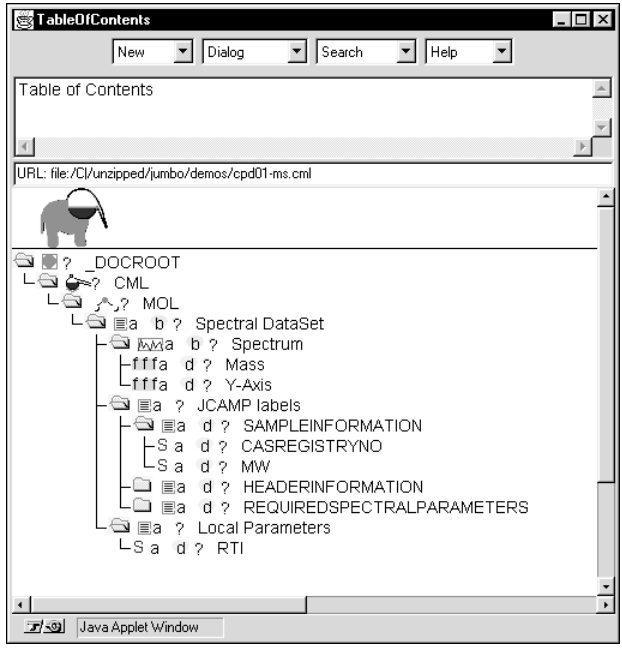


Рис. 1.9. Окно XML-браузера JUMBO

Виды деятельности, которые поддерживает OFX, включают банковские операции для потребителей и малого бизнеса, оплату по счетам потребителей и мелких бизнесменов, предъявление счетов к оплате, инвестиции, такие как акции, облигации, фонды взаимного кредитования. Во время написания книги был распространен OFX версии 1.5. Ожидается, что язык открытого финансового обмена будет способствовать развитию финансового обслуживания в реальном времени. Информацию об OFX можно найти на Web-сайте:

http://www.ofx.net/ofx/ab_main.asp

Заключение

В этой главе были рассмотрены основные фрагменты, складывание которых и образует многоцветную мозаику XML. Тем самым читатели получили базовые знания для работы над последующими главам, где будут детально описаны каждая из составляющих частей XML, что позволит затем перейти к созданию более сложных XML-приложений. Мы познакомимся со следующими вопросами:

- правильными и состоятельными документами;
- определением типа документа;
- описанием типа документа;
- каскадными таблицами стилей;
- расширяемыми языками таблиц стилей;
- анализаторами;

- пространствами имен;
- XML-схемами;
- просмотром XML-документов;
- некоторыми существующими XML-приложениями.

Надеемся, вы уже убедились в том, что язык XML имеет ряд серьезных преимуществ по сравнению с другими языками разметки. Несмотря на то, что XML еще не устоялся, в последующих главах вы получите прочную основу для создания XML-приложений. Читая последние главы, вы поймете, что сейчас самое время начинать писать на XML, а примеры продемонстрируют вам, что можно разрабатывать действующие XML-приложения, несмотря на чувство некоторой нестабильности, возникающее из-за непрерывной эволюции этого нового языка.

Глава 2. Правильные и состоятельные документы

В этой главе мы рассмотрим, каким образом создаются правильные и состоятельные XML-документы, соответствующие спецификации XML 1.0. Различие между ними состоит в том, что состоятельные документы имеют *определение типа документа* (DTD) и обязаны ему соответствовать, а правильные должны подчиняться куда более узкому набору нормативов. Обсуждение вопросов, связанных с синтаксисом, применяемым в подобных документах, поможет читателям получить базовую информацию, необходимую для самостоятельного создания XML-документов.

Правильные документы

Какие свойства, присущие разметке XML, необходимы для создания *правильного* (well-formed) документа? И что означает *правильность* документа?

Прежде всего следует еще раз напомнить, что синтаксис XML определяется спецификацией XML 1.0. Если программист разбирается в этом вопросе, то есть владеет спецификацией, он в состоянии сконструировать программу, которая сможет просмотреть документ, предположительно являющийся XML-документом, сравнить его и, если материал соответствует требованиям данного языка разметки, продолжить его дальнейшую обработку. Главная идея, лежащая в основе перечня условий XML, состоит в следующем: документы, называемые XML-документами, в определенном, точно сформулированном выше смысле должны быть доступны для понимания людей либо обрабатывающих приложений. Таким образом, правильным будет считаться документ, соответствующий минимальному набору требований, зафиксированных в описании языка. Соответствие этому минимальному набору является критерием, выполнение которого означает, что данный документ может рассматриваться как XML-документ. Этот набор включает в себя требования, обеспечивающие использование терминов языка правильным образом, и специальные указания, гарантирующие логическую связность документа в соответствии с определениями, данными в спецификации XML. Если документ не соответствует хотя бы одному из требований набора, считается, что произошла *неисправимая ошибка* (fatal error). В этом случае процессор *обязан* остановиться и информировать использующее его приложение об ошибке.

Состоятельные (valid) документы располагаются ступенькой выше правильных. Состоятельный документ по определению является правильным. Но состоятельные XML-документы еще и подчиняются стандартам, принятым для SGML-документов и могут быть использованы в качестве последних, а просто правильные документы – нет.

Принципиальная особенность состоятельного документа состоит в том, что он имеет *определение типа документа* (о нем мы поговорим во второй части главы) и ему соответствует. При этом в определении типа документа содержатся описания, которые обуславливают как структуру документа в целом, так и разрешенные *типы* (types) значений содержания данных.

Из сказанного следует, что правильные XML-документы не обязательно соответствуют стандарту SGML, поскольку могут и не иметь определения типа документа (DTD).

Верификация (validation) документа, то есть проверка его на состоятельность, включает контроль над тем, чтобы данные, содержащиеся внутри документа, могли быть переданы использующему их приложению; при этом также проверяются значения сегментов данных, величины которых не могут находиться вне приемлемых или ожидаемых пределов. Проверка на состоятельность в XML (то есть проверка соответствия документа и DTD) реально включает в себя только те механизмы, которые контролируют правильность типов значений. Однако необходимо учитывать, что возможности проверки, уточняющей, насколько значения попадают в некоторый требуемый интервал, весьма ограничены. Об этом мы также поговорим во второй части данной главы.

Сейчас будут рассмотрены только те вопросы, которые относятся к созданию правильного документа. При этом мы только вскользь коснемся употребляемых здесь понятий и терминов. Дело в том, что невозможно оторвать рассмотрение состоятельных документов от описания правильных, поскольку первые тоже обязаны подчиняться тем же ограничениям, что и вторые. Тем не менее, в начале главы мы не станем разбирать описания, которые необходимо поместить в DTD для обеспечения состоятельности документа.

На схеме (рис. 2.1) показано несколько описаний, которые должны находиться внутри раздела DTD. При этом следует учитывать, что в их число включены только те условия, выполнение которых обязательно, чтобы документ удовлетворял основным требованиям правильности.

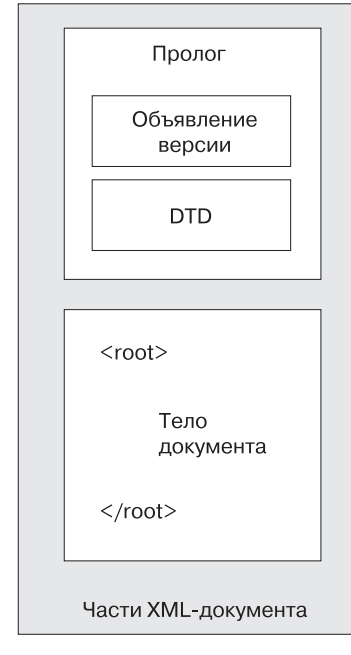


Рис. 2.1. Структура XML-документа

Приступаем к созданию документа

На самом деле объявление XML не является необходимым, однако спецификация, утвержденная консорциумом W3C, предполагает включать его для указания версии, выбранной для построения документа и в соответствии с которой следует употреблять тот или иной анализатор или процесс анализа. Если в документ включается объявление XML, оно должно быть помещено в самом его начале без предшествующих символов, даже без пробела.

Основной формат для объявления XML тот

же, что и для *команд приложений* (processing instructions): `<?name...?>`; позже мы подробней остановимся на этом вопросе. Команда приложения содержит его имя, за которым следуют собственно команды, передаваемые приложению. Объявление XML вводится для того, чтобы информировать анализаторы и другие приложения, каким способом следует обрабатывать данные в файле (документе XML). Минимальное содержание объявления XML состоит из зарезервированного имени `xml` и номера версии. При подготовке книги была использована версия 1.0, что одновременно указывает на номер спецификации языка XML, которой соответствует документ. Таким образом, начало нашего документа выглядит так:

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

Внутри объявления XML за номером версии могут следовать описание кодирования (необязательно) и объявление автономности (тоже необязательно), о которых мы поговорим во второй части главы. При этом необходимым является только номер версии.

Элементы

Во введении было сказано, что XML-документ в основном состоит из данных, размеченных с помощью тэгов. Каждая пара – начальный тэг и конечный тэг с находящимися внутри данными – составляет элемент.

```
<mytag>Здесь находятся какие-то текстовые данные</mytag>
```

Имена в начальном и конечном тэгах элемента *должны* быть одни и те же. Поскольку XML различает заглавные и строчные буквы, необходимо следить за тем, чтобы в именах они совпадали. Например, `<mytag>` и `<Mytag>` являются в XML двумя различными именами.

Как было отмечено во введении, текст обычно состоит из записанных особыми символами данных и разметки. Спецификация XML 1.0 определяет предложение между тэгами `"Здесь находятся какие-то текстовые данные"` как *символьные данные* (character data). Два тэга, находящиеся по обе стороны от символьных данных, образуют *разметку* (markup).

В настоящий момент читателям достаточно знать, что символьные данные, находящиеся между тэгами элементов, могут состоять из любой последовательности допустимых символов (соответствующих стандарту Unicode, который мы обсудим позже), кроме символа `<`, открывающего элемент. Символ `<` запрещен, поскольку анализатор может принять его за начало нового тэга и выдаст ошибку, когда не найдет имя тэга, которое, в соответствии с правилами, должно следовать за этим символом.

Уточнение *Возникает вопрос, почему нельзя написать программу, способную обрабатывать и угловую скобку? Напомним, что одна из главных целей разработки данной спецификации XML заключается в том, чтобы помочь пользователям сравнительно легко справляться с созданием программы для обработки XML-документов. Если с самого начала не отвлекать внимание пользователя дополнительными проблемами, написать приложение-анализатор будет гораздо проще.*

Перед именем конечные тэги содержат символ «косая черта», `</tagname>`, чтобы отличить их от начальных тэгов, `<tagname>`. Однако для элемента, не имеющего содержания, существует сокращенное обозначение. Оно состоит из одного тэга с символом косой черты после имени, `<my_empty_element/>`. Пока не стоит задумываться над тем, как используются эти пустые тэги, к ним мы вернемся позже; сейчас просто обратите внимание на синтаксис.

Вы, надеюсь, заметили, как в предыдущем примере были названы тэги? Точно и весьма незамысловато – `mytag` (мой тэг). XML позволяет давать тэгам любые имена, поэтому вы можете создать следующий абсолютно правильный тэг:

```
<Wop-bop-a-loo-bop-a-lop-bam-boom_Tutti-frutti>
   As sung by Little Richard
</Wop-bop-a-loo-bop-a-lop-bam-boom_Tutti-frutti>
```

Совет *Хотя в спецификации XML не упоминается ни о чем таком, что бы напоминало категоричные запреты, некоторые программы-анализаторы могут устанавливать ограничения на длину имен. Поэтому прежде, чем браться за особо размашистые имена тэгов, загляните в руководство к программе.*

Правильные имена начинаются с буквы, символа подчеркивания, или двоеточия, за которыми может следовать любая комбинация букв, цифр, дефисов, символов подчеркивания, двоеточий или точек. Имена не должны начинаться с трех символов `xml` в любом сочетании заглавных и строчных букв, поскольку эта комбинация букв зарезервирована для использования в стандартных командах XML. Не рекомендуется также начинать имена с двоеточия, поскольку позже, когда вы возьметесь за создание более сложных документов (в частности, с использованием пространств имен, о чем говорится в главе 4), это может вызвать путаницу.

В произвольном выборе имен тэгов заключается вся сила XML. Вы можете дать тэгам имена, которые *описывают* содержание элемента. Предположим, что мы создаем таксономическую базу данных по дикой природе Южной Америки. Общая запись для горных лам могла бы выглядеть так:

```
<taxonomy>
   <class>Mammalia
      <order>Artiodactyla
         <suborder>Tylopoda
            <family>Camelidae
               <genus>Lama</genus>
            </family>
         </suborder>
      </order>
   </class>
</taxonomy>
```

Вот перед вами пример правильного документа. Для создания подобных материалов все или почти все, что следует знать о применении разметки (в соответствии со спецификацией консорциума W3C) заключено в трех простых правилах:

- в документе может содержаться один или большее число элементов;
- документ должен включать в себя один элемент с уникальным именем, никакая часть которого не содержится внутри никакого другого элемента. Этот элемент называется *корневым элементом* (root element);
- все другие элементы внутри корневого элемента должны быть вложены правильно.

Согласно этим правилам, следующие три документа являются правильными:

```
<!-- Первый документ. -->
<my_document></my_document>
```

```
<!-- Второй документ. -->
<?xml version="1.0"?>
<class>Mammalia</class>
<!-- Третий документ.-->
<root>
   <class>Mammalia</class>
</root>
```

Строго говоря, даже такой «документ» является правильным:

```
<emptytag/>
```

Чтобы тэги были вложены правильно, в структуре документа не должно быть перекрытий между начальным и конечным тэгами двух разных элементов. То есть допустима следующая конструкция:

```
<parent>
   <child>Некоторая информация</child>
</parent>
```

Здесь дочерний элемент `<child>` начинается и заканчивается внутри родительского элемента `<parent>`. А вот приведенная ниже конструкция недопустима:

```
<badparent>
   <naughtychild>
       Некоторая текстовая информация
<badparent>
   </naughtychild>
```

В этой конструкции элемент `naughtychild` выходит за пределы элемента `badparent`, который на самом деле должен полностью содержать в себе элемент `naughtychild`.

Таким образом, теперь можно попробовать сконструировать правильный документ, который завершает запись таксономической базы данных для обыкновенной ламы:

```
<taxonomy>
   <class>Mammalia
       <order>Artiodactyla
           <suborder>Tylopoda
               <family>Camelidae
                   <genus>Lama
```

```
                    <species>glama
                        <common_name>
                            Llama
                        </common_name>
                    </species>
                    <species>pacos
                        <common_name>
                            Alpaca
                        </common_name>
                    </species>
                    <species>guanicoe
                        <common_name>
                            Guanaco
                        </common_name>
                    </species>
                    <species>vicugna
                        <common_name>
                            Vicuna
                        </common_name>
                    </species>
                </genus>
            </family>
        </suborder>
    </order>
</class>
</taxonomy>
```

Заметим, что элемент <species> содержит как символьные данные, так и дочерний элемент <common_name>, а элемент <genus> включает в себя как символьные данные, так и элементы <species>. Такой порядок вложения элементов совместно с данными допустим, так как в обоих случаях дочерние элементы начинаются и кончаются внутри содержащего их элемента.

Элементы, вложенные указанным выше способом, позволяют представить структуру соотношений различных частей информации в целом. В полной таксономической базе данных у нас, конечно же, будет храниться более, чем один-единственный класс, например Mammalia, Chrondrichthyes, Myxini..., но каждый из этих классов в базе данных относиться к одному и тому же уровню. Классы делятся на отряды, те, в свою очередь, на подотряды. Вложенность, таким образом, является мощным средством описания подобных соотношений.

Законченная структура таксономической базы данных для ламы может выглядеть примерно так:

```
<taxonomy>
    <class>Mammalia
        <order>...</order>
        <order>...</order>
        ...
    </class>
    <class>Chondrichthyes
        <order>...</order>
```

```
        <order>...</order>
        <order>...</order>
        ...
    </class>
    <class>Myxini
    ...
    </class>
</taxonomy>
```

Атрибуты

Элементы могут иметь атрибуты. Атрибуты представляют собой значения, которые передаются приложению анализатором XML, но не составляют часть содержания элемента. Атрибуты вводятся как часть начального тэга элемента. Рассмотрим пример:

```
<trousers areJeans = "No">
   Sportif multi-coloured spandex flares
</trousers>
```

Здесь строка `areJeans` («являются джинсами») представляет собой атрибут тэга `trousers` (брюки). Для данного элемента `trousers` атрибуту `areJeans` присваивается значение `"No"`. Атрибуты также могут быть частью пустого элемента. Короче говоря, может существовать следующий пустой элемент:

```
<has_surfboard cansurfwell = "Yes"/>
```

Элементы могут иметь любое количество атрибутов:

```
<BigElement attrib_1="one" attrib_2="two" attrib_3="three" ... />
```

Для того чтобы документ считался правильным, имя атрибута не должно повторяться в том же самом начальном тэге элемента. Таким образом, следующие тэги недопустимы, потому что атрибут `NastyCloneAttrib` повторяется дважды:

```
<BadElement NastycloneAttrib="ugh!" NastycloneAttrib="sicky!">
   Blah blah blah
</BadElement>

<BadEmptyElement NastycloneAttrib="pew!" NastycloneAttrib="yet more ugh!"/>
```

Для документа, являющегося правильным, но не состоятельным, описаний разметки в DTD, указывающих на то, какие типы данных могут содержать значения атрибутов, не существует (так как, строго говоря, такие документы просто не имеют DTD). В этом случае все значения атрибутов будут рассматриваться как данные типа CDATA (обычные символьные данные). Величины атрибутов, представляющие собой строку, заключенную в кавычки, как, например, строка `pew!` в предыдущем случае, не должны содержать символов `<`, `&` или непарных вхождений одинарных и двойных кавычек.

Существуют дополнительные ограничения на допустимые значения правильных атрибутов, но эти ограничения касаются только компонентов.

Компоненты

Детально *компоненты* (entities) будут рассмотрены во второй части главы, посвященной состоятельным документам, однако чтобы до конца разобраться с правильными документами, о них следует упомянуть и в этом параграфе.

Существует два типа компонентов: *обычные компоненты* (general entities) и *параметрические компоненты* (parameter entities).

Компоненты, как правило, используются внутри документа для того, чтобы избежать многократного набора больших участков текста. Они обеспечивают работу механизма, с помощью которого можно соотнести некоторое имя с неким большим участком текста, и затем просто вставить это имя в том месте в документе, где нужно поместить данный текст. При обработке документа имя компонента замещается соответствующим текстом, и, таким образом, экономятся и время, и силы. Если появится необходимость что-то изменить, поправки достаточно внести один раз, во вставляемый участок текста, а не многократно по всему документу.

Все компоненты состоят из двух частей: *описания* (declaration) и *ссылки на компонент* (entity reference). К ссылке на компонент мы еще вернемся, а пока начнем с описания.

Обычные компоненты

Описание обычных компонентов имеет следующую форму:

```
<!ENTITY entityname "Некоторый текст замены" >
```

для *внутреннего компонента* (internal entity) или

```
<!ENTITY entityname SYSTEM "http://www.someserwer.com/dir/somefile.xml" >
```

для *внешнего компонента* (external entity).

Для внешнего компонента, в отличие от внутреннего, содержание вставляемого текста находится в отдельном текстовом файле, а не в XML-документе. Ключевое слово `SYSTEM` указывает анализатору, что файл надо искать по определенному адресу, указанному в виде *унифицированного идентификатора ресурсов* (Uniform Resource Identifier, URI), в данном случае по адресу http://www.someserver.com/dir/somefile.xml. Заметим также, что разделение компонентов на внешние и внутренние указывает только на то, где находится *текст замены* (replacement text): внутри описания компонента или во внешнем файле. Эта классификация не дает информации о том, где помещается само описание компонента: внутри данного документа или нет. Поэтому в данном контексте слова «внутренний» и «внешний» лучше понимать как типы компонентов, а не как местонахождение описания или его частей.

Отсюда следует, что можно определить внутренний компонент в файле, находящемся вне данного документа. Точно так же внешний компонент может быть описан внутри документа. Эти детали приобретают важное значение при изучении состоятельных документов, к которым мы вернемся во второй части главы.

Авторы и редакторы спецификации XML при обсуждении компонентов и их характеристик пришли к выводу, что полезно провести различие между *символьным*

значением (literal value) компонента в описании и *текстом замены* (replacement text), получаемым после раскрытия ссылки на компонент. В примере, приведенном выше, текстовая строка `"некоторый текст замены"` в описании представляет собой символьное значение компонента, а та же самая строка после обработки и раскрытия ссылки на компонент уже является текстом замены.

Параметрические компоненты

Во второй части главы будут детально рассмотрены *параметрические компоненты* (parameter entities), однако уже сейчас необходимо дать краткое определение этому понятию. У параметрических компонентов особое назначение, при этом их описания выглядят так же, как и описания обычных компонентов, за исключением того, что они содержат дополнительный символ `%`:

```
<!ENTITY % entityname "Некоторый текст замены" >
```

Параметрические компоненты тоже могут быть внутренними и внешними.

Описания компонентов и декларация <!DOCTYPE [...]>

Декларация `<!DOCTYPE [...]>` следует в прологе документа сразу за объявлением XML и используется для того, чтобы обозначить раздел, в котором содержатся все необходимые документу описания. Если возникает необходимость описывать компоненты, то декларация `DOCTYPE` тоже включается в правильный документ. Очень простой XML-документ, содержащий подобную декларацию, мог бы иметь следующий синтаксис:

```
<?xml version="1.0" ?>
<!DOCTYPE mydocname [
...здесь находятся описания...
]>

<mydocname></mydocname>
```

Обратите внимание, что в правильных документах все описания компонентов должны находится внутри квадратных скобок декларации `<!DOCTYPE [...]>`, расположенных в начале документа. Это условие сильно упрощено, поскольку *физическая* и *логическая* (или синтаксическая) позиции описания внутри данного документа не обязательно совпадают. Описания, на которые имеются ссылки внутри декларации `DOCTYPE`, на самом деле могут находиться во внешнем файле. При этом следует иметь в виду: логически эти описания все равно находятся внутри декларации `DOCTYPE`, несмотря на то, что они вынесены во внешний файл.

Описания, которые находятся внутри декларации `DOCTYPE`, а также физически внутри документа, называются *внутренним подмножеством* (internal subset) определения типа документа. Те описания, которые логически находятся внутри описания `DOCTYPE`, но являются внешними по отношению к документу, называются *внешним подмножеством* (external subset) определения типа документа.

Таким образом, соглашение о том, что все описания компонента должны находиться внутри квадратных скобок описания `DOCTYPE`, в своей основе правильно, однако с выполнением этого условия возникают некоторые сложности. Прежде всего это касается различий между способами, с помощью которых верифицирующие

и неверифицирующие анализаторы (то есть те, которые лишь проверяют, является ли документ правильным) обрабатывают документ. Несмотря на то, что некоторые описания могут находиться внутри DTD только логически, но не физически, неверифицирующие анализаторы при обработке описаний компонентов преимущественно обращают внимание на физическое местонахождение компонентов или ссылок на них. Неверифицирующие анализаторы не обязаны читать какие-либо описания компонентов, находящиеся вне документа, поэтому, если достаточно, чтобы документ был только правильным, имеет смысл поместить все описания компонентов во внутреннее подмножество.

Совет *Выбор для реализации проекта просто правильных документов, содержащих обычные компоненты, и использование неверифицирующих анализаторов подразумевает, что сами проекты и множества компонентов в документах сравнительно невелики по объему. В противном случае затраты (времени, денег, усилий) на вставку всех действующих описаний компонентов в каждый из документов скоро окажутся недопустимо большими. Переход к состоятельным документам и верифицирующим анализаторам дает возможность с большей легкостью перемасштабировать задачу в крупный проект, позволяя переносить описания компонентов во внешние файлы, на которые может ссылаться каждый отдельный документ. И наоборот, если желательно использовать преимущество быстрой обработки правильных документов (при условии, что это удовлетворяет вашим требованиям) неверифицирующим анализатором, то вообще не следует использовать в документах компоненты.*

Со временем, в этой же главе, мы подробнее рассмотрим описание `DOCTYPE`, а в данный момент следует запомнить лишь следующий синтаксис:

```
<!DOCTYPE mydocname [
<!ENTITY entityname "Некоторый текст замены">
]>

<mydocname></mydocname>
```

Уточнение *Заметим, что имя документа должно быть точно таким же, как и имя его корневого элемента, какое бы имя вы ни подставили вместо обозначения mydocname, приведенного в уже указанном выше примере.*

Имя компонента может быть каким угодно, лишь бы оно соответствовало правилам построения имен, разобранным ранее, в разделе «Элементы». Если компонент является внешним, он должен содержать только текст (символьные данные и разметку) или символьные данные.

Ссылки на компоненты

После того как описание создано, на компонент внутри документа можно сослаться, используя его имя в следующем виде:

```
&entityname;
```

Совет *При просмотре XML-файла анализатору необходимо прочитать описание компонента прежде, чем появится любая ссылка на этот компонент; в противном случае он тут же выдаст ошибку. Ссылки внутри документа на компоненты, вообще не описанные в нем, также запрещены (и попросту бесполезны).*

Между символом &, именем компонента и замыкающей точкой с запятой не должно быть пробелов. Следующие ссылки из-за вставленных пробелов не работают:

```
& entityname;
&entityname ;
& entityname ;
```

В случае использования обычного компонента ссылка на компонент обычно дается внутри текстового содержания элемента:

```
<myElement>I am reminded of the Gettysburg address in
   which Abe said, "&gettysburg;"
</myElement>
```

Если требуется неоднократно включать в документ один и тот же отрывок текста (например, заявление об авторских правах в конце каждой страницы), этот механизм подстановки может оказаться очень полезным.

Неверифицирующие анализаторы только проверяют внутренние компоненты на правильность и не обязаны читать и включать в документ содержание внешних компонентов, тем более контролировать, правильны ли они. Поэтому в документ, удовлетворяющий минимальным требованиям правильности, представляется разумным включать только описания внутренних компонентов и ссылки на них. Это обеспечит желаемую форму данных, передаваемых приложению после обработки анализатором.

В то время как на внутренние компоненты можно ссылаться внутри значения атрибута элемента, нельзя подобным же образом делать ссылки на внешние компоненты. Причина кроется в том, что внешние компоненты могут представлять собой символы во всевозможных кодировках, и помещение данных внутри значения атрибута в кодировке, отличной от кодировки основного документа, рассматривается как лишняя трудность. Если бы такое обращение было разрешено, анализаторы стали бы существенно сложнее.

Уточнение *Ссылки на обычные компоненты не могут находиться внутри описания DOCTYPE.*

Заметим также, что все внутренние или внешние компоненты, описанные способом, о котором было сказано выше, в спецификации XML 1.0 называются *разбираемыми компонентами* (parsed entities). Такие компоненты по определению могут содержать только символьные данные или разметку XML.

Текст замены для всех компонентов должен быть правильным и в полной мере соответствовать требованиям спецификации XML. Это означает, что ни-

какой начальный тэг, конечный тэг, элемент или ссылка на компонент не могут начинаться в одном компоненте и заканчиваться в другом. Например, *недопустимо* следующее:

```
<!ENTITY myent "<mytag>Текст элемента.." >
```

И последнее: компоненты не должны ссылаться на самих себя ни прямо, ни косвенно. Поэтому запрещены такие описания:

```
<ENTITY net "Сведения о &net;">
<ENTITY one "ссылка на &two;">
<ENTITY two "ссылка на &one;">
```

Компоненты могут ссылаться на другие компоненты, если это не приводит к прямой или косвенной ссылке на самих себя, как показано выше. В соответствии с этим разрешены следующие ссылки:

```
<ENTITY where      "на коврике">
<ENTITY didwhat    "сидела &where;">
<ENTITY who        "Кошка &didwhat;">
```

Если затем использовать ссылку `&who;` внутри содержания элемента, анализатор развернет ссылки на компоненты, и сформирует текст замены «Кошка сидела на коврике».

Ссылки на параметрические компоненты

Ссылки на параметрические компоненты записываются с использованием следующего синтаксиса:

```
%param_ent;
```

Уточнение *В рамках внутреннего подмножества описаний ссылки на параметрические компоненты могут появляться только между другими описаниями подмножества, но не могут быть использованы внутри описаний.*

Во внешнем подмножестве ссылки на параметрические компоненты могут быть использованы *внутри* других описаний.

Более подробно мы обсудим параметрические компоненты в разделах, посвященных состоятельным документам, так как в правильных документах в них нет особой нужды. Причина состоит в следующем:

- несмотря на то, что неверифицирующие анализаторы обрабатывают любую информацию внутри декларации `DOCTYPE`, они не обязаны считывать и раскрывать внешние параметрические компоненты и ссылки на них. В правильных документах следует избегать использования внешних параметрических компонентов;
- логическая форма внутренних параметрических компонентов и возможность ссылаться на них только в описаниях внутреннего подмножества, означает, что компоненты подобного рода фактически никак нельзя использовать в правильных, но не являющихся состоятельными документах. (Это станет понятно позже, когда мы обсудим параметрические компоненты в контексте состоятельных документов.)

Ссылки на компоненты внутри значений атрибутов

Ранее, во время обсуждения атрибутов элементов, мы видели, что среди значений, которые могут принимать атрибуты, присутствуют символьные данные. Соблюдая некоторые ключевые ограничения, можно также включать в значения атрибутов ссылки на обычные компоненты. Например, вы можете внести в документ, относящийся к области зоологии, элемент, ссылающийся на описание ДНК ламы, который выглядит следующим образом:

```
<Llama DNAsequence="&llama_DNA;">
```

Однако здесь тоже существуют определенные ограничения, на которые обязательно следует обратить внимание. Прежде всего, значение атрибута не может прямо или косвенно содержать ссылку на *внешний* компонент. Поэтому компонент `llama_DNA;` не имеет права быть внешним, поскольку в этом случае ссылка на него в значении атрибута в указанном выше примере была бы прямой ссылкой на внешний компонент.

Предположим теперь, что компонент `&llama_DNA;` ссылается на внутренний компонент, который мы описали ранее:

```
<!ENTITY llama_DNA "CAGTCAGTCAGT- &base_sequence;">
```

причем `&base_sequence;` (основная последовательность) представляет собой ссылку на внешний компонент, описанный как

```
<!ENTITY base_sequence SYSTEM "http://www/someserver.com/dir/bases.txt;">
```

В этом случае ссылка на компонент `llama_DNA`, содержащаяся в значении атрибута `DNAsequence`, имеет ссылку на внешний компонент и, следовательно, использование `&llama_DNA;` в качестве значения `DNAsequence` представляет собой косвенную ссылку на внешний компонент, что запрещено правилами языка.

Следующее ограничение на значения атрибутов в правильных документах касается текста замены любого компонента, на который существует прямая или косвенная ссылка в значении атрибута. Подобная запись не должна содержать символ `<` (меньше чем). (Существует единственное исключение, которое будет разобрано чуть позже).

Для чего необходимо вводить это ограничение? Предположим, что текст замены внутреннего компонента `llama_DNA` содержит нечто вроде `"blah blah CAGT<CAGTCAGT"`, в этом случае анализатор вставит этот отрывок в элемент `<Llama>` и получит следующее:

```
<Llama DNAsequence="blah blah CAGT<CAGTCAGT">
```

Верифицирующий анализатор, проверяющий документ на правильность, интерпретирует присутствие символа `<` как начало нового тэга, и выдаст ошибку, решив, что этот новый тэг вложен неправильно (не правда ли, грубая ошибка?).

Единственное исключение из этого правила состоит в том, что можно включить в значение атрибута ссылку на предопределенный компонент `<`, имеющий в качестве текста замены символ `<` (меньше чем). В следующем подразделе мы увидим, как работает это исключение.

Символьные компоненты

Ссылки на символы (character references) также представляют собой ссылки на компоненты. Они используются в двух случаях:

- чтобы создавать ссылки на символы, принадлежащие множеству символов ISO/IEC 10646 (см. далее, в подразделе «Ссылки на символы»), но не доступные для ввода непосредственно с клавиатуры или из операционной системы;
- чтобы до передачи данных в приложение клиента предупредить обработку анализатором *управляющих символов* (escape characters), которые могут быть приняты за разметку.

Уточнение *Множество символов ISO/IEC 10646 представляет собой стандарт Международной организации по стандартам (International Standards Organization, ISO), определяющий способ, с помощью которого некоторые символы (например, буквы в написанных здесь словах) закодированы в двоичном формате на компьютере. Каждой букве или символу соответствует уникальный двоичный код.*

Ссылки на символы

В первом случае ссылки на символы имеют вид:

- `&#decimal_character_code;` (десятичный код символа);
- `&#hexadecimal_character_code;` (шестнадцатеричный код символа),

где код символа представляет собой номер символа в соответствии со стандартом ISO/IEC 10646. Стандарт кодирования символов Unicode является эквивалентом стандарта ISO. Во время подготовки книги к изданию кодировку символов можно было найти по адресу ftp://ftp.unicode.org/Public/UNIDATA/UnicodeData-Latest.txt.

Вот примеры кодов символов для согласной «tt», употребляемой в бенгальском языке:

- `&#x099F;` в шестнадцатеричной кодировке;
- `&#2463;` в десятичной кодировке.

Отложенный разбор символов

Со вторым случаем мы встречаемся, когда в качестве символьных данных необходимо использовать знаки, которые нельзя записывать в их литерном представлении, поскольку при разборе анализатор может выдать ошибку. Спецификация XML указывает для правильных документов некоторое множество предопределенных символьных компонентов, которые при необходимости используются вместо литерного представления символов. В правильных документах, в отличие от состоятельных, эти символьные компоненты не обязательно описывать в явном виде. Однако мы все-таки рассмотрим такую операцию, чтобы понять, как именно работают символьные компоненты в правильном документе. Ниже приведены описания, неявно включаемые в каждый правильный документ и содержащие коды в соответствии со стандартом кодирования Unicode для символов, употребления

литерной формы которых следует избегать. (Коды ASCII символов соответствуют первым 256 кодам стандарта Unicode.)

```
<!ENTITY   lt          "&#38;#60;">
<!ENTITY   gt          "&#40;">
<!ENTITY   amp         "&#38;#38;">
<!ENTITY   apos        "&#39;">
<!ENTITY   quot        "&#34;">
```

В табл. 2.1 приведены символы, с помощью которых ссылки после разбора анализатором заменяются на компонент.

Таблица 2.1. Предопределенные символьные ссылки языка XML

Имя компонента	Название символа	Вставляемый символ
&lt;	меньше чем	<
&gt;	больше чем	>
&amp;	амперсанд	&
&quot;	кавычки	"
&apos	апостроф	'

Легко заметить, что в описаниях символов < (меньше чем) и & (амперсанд) даны двойные ссылки на символы, например `"&#38;#60;"` для компонента `lt`. Это означает, что данные управляющие символы – двойные, и их разбор следует отложить. Причина в том, что эти символы разрешается употреблять в литерной форме, то есть как < и &, только тогда, когда они являются разделителями разметки. Символ < используется в тэгах, например <tag>, включая и те случаи, когда он встречается в начале описания компонентов; амперсанд & используется в ссылках на компоненты, `&ent;`. Поскольку ссылки на символы в описываемых компонентах раскрываются сразу (`&#38;` меняется на `&`), текстом замены для компонента `lt` является `&#60;`. При разборе он по ссылке `<` был бы заменен символом <. Если бы мы не сослались на компонент `lt` указанным выше способом, значением компонента стал бы символ <, и при проверке документа на правильность анализатор сообщил бы об ошибке.

Из этого правила существуют два исключения:

- символы & и < могут встречаться в литерной форме в комментариях, командах приложений XML и секциях CDATA, о которых поговорим позже. Сейчас только следует запомнить: это исключение действует потому, что анализатор игнорирует содержание указанных компонентов во время разбора документа;
- символы & и < могут встречаться в буквенном значении компонента (во вставляемом тексте) при описании внутреннего компонента.

Необходимое замечание: в случае второго исключения *не рекомендуется* делать следующее:

```
<!ENTITY   ampsign     "&" >
<!ENTITY   ltsign      "<" >
```

Дело в том, что при *записи* описаний разрешается вставлять символы `&` или `<` в буквенное значение компонента только потому, что в этот момент символы на самом деле не рассматриваются как разделители разметки. Однако при ссылке на компонент эта операция не будет соответствовать условиям правильности документа, так как символ будет рассматриваться как разметка, и правила будут нарушены. В этом случае неверифицирующий анализатор отвергнет документ.

Отложенный разбор символа решает проблему. Вопрос в том, как работает эта процедура и зачем она нужна? Чтобы на него ответить, обратимся к примеру, приведенному в «Приложении D» к спецификации XML 1.0. В документе использовано следующее описание:

```
<!ENTITY example "<p> Мы можем отложить разбор амперсанда (&#38;#38;) численно
(&#38;#38;#38) или с помощью обычного компонента (&amp;amp;). </p>" >
```

Когда XML-процессор обрабатывает это описание, он распознает ссылки на символы и заменяет их, получая следующую строку, которая затем сохраняется как значение компонента `example`.

```
<p> Мы можем отложить разбор амперсанда (&#38;) численно (&#38;#38;) или с помощью
обычного компонента (&amp;amp;). </p>
```

Если документ содержит ссылку на компонент `&example;`, то этот текст будет разобран повторно и, как говорят авторы спецификации, начальные и конечные тэги элемента `<p>` будут распознаны, а три ссылки (на компоненты) распознаны и раскрыты; в результате появится элемент со следующим содержанием (только данные, никаких разделителей или разметки):

```
Мы можем отложить разбор амперсанда (&) численно (&#38;) или с помощью обычного
компонента (&amp;).
```

Вы можете спросить, почему для получения текста, содержащего `&#38;`, приходится вводить в описание компонента `&#38;#38;#38;`, а если необходимо выявить `&`, достаточно вставить `&amp;`.

Причина в том, что спецификация XML определяет набор правил, которые указывают анализатору, как следует обрабатывать компоненты в зависимости от их контекста внутри документа. В разделе 4.4 спецификации помещена таблица применения правил. Ниже, в табл. 2.2, воспроизведены три столбца этой таблицы. В пропущенных столбцах рассматриваются внешние неразбираемые компоненты, которые могут встречаться только в состоятельных документах и параметрических компонентах.

Таблица демонстрирует способы, с помощью которых XML-анализатор обрабатывает компоненты во всевозможных контекстах внутри документа.

Если обратиться ко второму столбцу таблицы, описывающему разбор внутренних компонентов, можно обнаружить, что неверифицирующий анализатор допускает следующую конструкцию:

```
<?xml version="1.0" ?>
<!DOCTYPE col2 [
<!ENTITY ent "textEntity" >
```

```
<!ENTITY otherent "yes" >

<!ENTITY doodah "Big Bob's &amp;ent" >
]>
<col2 attrib="&otherent;"> some text including &ent; or &doodah;</col2>
```

После ее обработки получаем:

```
<col2 attrib="yes"> some text including textEntity or Big Bob's &ent; </col2>
```

Таблица 2.2. Правила разбора компонентов в зависимости от контекста

Контекст	Тип объекта		Символьный
	Внутренний, обычный	Внешний обычный, разбираемый	
Ссылка в содержании	Включается	Включается при разборе верифицирующим анализатором	Включается
Ссылка в значении атрибута	Включается в литерном виде	Запрещен	Включается
Встречается как значение атрибута	Запрещен	Запрещен	Не распознается
Ссылка в значении компонента	Пропускается	Пропускается	Включается
Ссылка в DTD	Запрещен	Запрещен	Запрещен

Когда анализатор считывает описание, ссылка на символьный компонент `amp` им пропускается, поскольку это ссылка отнесена на внутренний обычный компонент внутри значения компонента `doodah`. Когда же ссылка на компонент `doodah` встречается внутри документа, подстановка символа амперсанда выполняется, но подстановка значения компонента `ent` не происходит, поскольку вставленный символ `&` автоматически рассматривается как символьные данные, а не разметка.

Примеры для третьего столбца можно пока не рассматривать, поскольку неверифицирующий анализатор не обязан включать текст замены внешнего разбираемого компонента.

Что касается четвертого столбца, стоит обсудить две возможности, при которых ссылки на символы не распознаются или запрещаются. Для значений атрибутов важно различать случай:

```
<Element    attrib="&#60;" />
```

где вставляемый символ включается во время разбора с целью ссылки на него в дальнейшем; и вариант:

```
<Element    BadRefattrib="#60;" />
```

где символьный компонент просто не распознается, поскольку не соответствует синтаксису ссылок на компонент.

В тех случаях, когда табл. 2.2 устанавливает, что ссылки на символьные компоненты в DTD не разрешены, это означает, что подобные ссылки на символы могут быть сделаны только внутри значения в виде символьной строки для описания компонента (или внутри значения атрибута для состоятельного документа).

Таким образом, следующая запись допустима,

```
<!ENTITY doodah   "text including &#38;textEntity;" >
```

а вот эта, со ссылкой на символ < вместо него самого, – нет:

```
&#60!ENTITY doodah "text including &#38;textEntity;" >
```

Кроме того, неправильным является употребление `&#60;` между описаниями ссылки.

Отложенный разбор участков данных

Теперь пришел черед обсудить различные методы отложенного разбора (escaping) определенных участков данных в документе. Для чего это нужно, выяснится по мере обсуждения того или иного способа.

Секции CDATA

Секции CDATA можно использовать в любом месте документа, где могут появиться символьные данные. Их цель – выполнение отложенного разбора блоков текста, содержащих символы, которые в противном случае могут быть приняты за разметку. Пусть окончательный вид нашего документа должен принять такую форму:

```
<DisplayedElem>
"Time for bed" said Zebedee
</DisplayedElem>
```

Как сделать, чтобы элемент `DisplayedElem` не рассматривался как разметка? Надо использовать секцию CDATA следующим образом:

```
<NotDisplayed>
<![CDATA[<DisplayedElem> "Time for bed" said Zebedee </DisplayedElem>]]
</NotDisplayed>
```

Комментарии

Профессиональное программирование подразумевает включение в код комментариев для объяснения, напоминания, просто указания на что-либо, выделения участков кода для упрощения его восприятия и интерпретации в будущем. В большинстве случаев возможность называть элементы языка XML по собственному усмотрению освобождает программиста от основного груза комментариев. Однако если имена тэгов не несут достаточной информации, в XML имеется механизм создания комментариев. Их синтаксис совпадает с синтаксисом комментариев в HTML:

```
<!-- Здесь можно поместить любой текст. -->
```

Текст комментариев не должен содержать дефис или последовательность двух дефисов, чтобы анализатор не перепутал их с окончанием комментария.

Комментарии могут располагаться за пределами других типов разметки или в допустимых местах внутри декларации `DOCTYPE`. Таким образом, вновь обратившись к одному из наших предыдущих примеров, комментарии в него можно вставить следующим способом:

```
<?xml version="1.0" ?>
      <!-- Документ, демонстрирующий вставку компонентов. -->
```

```
<!DOCTYPE col2 [
       <!-- Мы вошли внутрь описания DOCTYPE -->
<!ENTITY ent "textEntity" >
       <!-- и описали один компонент, -->
<!ENTITY otherent "yes" >
       <!-- затем другой, -->
<!ENTITY doodah "Big Bob's &amp;ent" >
]>
       <!-- а теперь мы выходим из описания DOCTYPE. -->
<col2 attrib="&otherent;"> некоторый текст, включающий &ent; или &doodah;
       <!-- Мы можем поместить комментарий даже здесь -->
</col2>
       <!-- и здесь. -->
```

Комментарии *не должны* находится внутри описаний и тэгов элементов. Иначе говоря, следующая запись недопустима:

```
<!ENTITY speech
   <!-- Нобелевская речь Боба Оскара. -->
"Я пользуюсь случаем поблагодарить моих родителей, мою сестру Фрэнки, моего
парикмахера ..." >
```

Недопустима и такая запись:

```
<Arnie     <!-- с угрозой -->>
   Я буду драться
</Arnie>
```

Теперь, наконец, можно перейти к следующему подразделу.

Команды приложения

Документы могут содержать *команды приложения* (processing instructions), использующие данные XML. Эти команды состоят из наименования приложения-адресата и его команд:

```
<?NameOfTargetApp Instructions for App ?>
```

Команды приложений не являются частью символьных данных. При передаче они должны сохранить первоначальную форму. Другими словами, анализатор должен разобрать документ и передать эти команды приложению, использующему анализатор, в неизменном виде. Команды приложений могут быть напрямую адресованы приложению, использующему анализатор, но могут также содержать команды другому приложению, которое вызывается обрабатывающим приложением. В таком случае команды записываются в формате того приложения, которому они адресованы. Имя приложения-адресата ни в коем случае не может быть сочетанием `xml` в любой комбинации строчных и прописных букв, поскольку имя `xml` зарезервировано для других целей.

Команды приложений могут располагаться в тех же местах документа, что и комментарии (см. пример предыдущего раздела «Комментарии»).

DTD: состоятельный документ

Читая книгу, руководство или журнальную статью, читатели редко обращают внимание на визуальное оформление материала. Если текст хорошо написан, композиционно выстроен, его структура прозрачна и не бросается в глаза. Мельком мы еще можем обратить внимание на заголовки и абзацы, однако многие другие стороны расположения и подачи печатного текста просто ускользают от нас. Тем не менее, всем понятно, что какой-то порядок безусловно необходим – при отсутствии структуры информацию просто невозможно передать. Структурирование делает ее доступной для понимания человеком или каким-либо приложением. Другой стороной членения информации является возможность *синтаксического разбора* (parsing) документа – например, можно определенным способом проверить наличие необходимых частей структуры и выдать сообщение об отсутствии любой из них.

Если структура начального документа спланирована умело, то и последующие документы, скорее всего, тоже будут логически завершенными, поскольку они по умолчанию обязаны соответствовать одним и тем же организационным принципам. Процесс *проверки на состоятельность* или *верификация* (validation) XML-документов состоит в проведении контроля за тем, является ли данный документ логически завершенным по отношению к некоторой предопределенной структуре (ее синтаксис, в свою очередь, определяется спецификацией, утвержденной консорциумом W3C). Этот принимаемый за образец способ организации материала, на соответствие с которым проверяется документ, содержится в DTD, или *определении типа документа*. (Обозначение DTD было представлено в первоначальной спецификации XML 1.0, в то время как XML-данные и DCD относятся к внесенным позже альтернативным способам создания схем данных для XML-документов.) DTD может находиться в том же файле, что и весь документ, или содержаться в отдельном файле, на который ссылается каждый из соответствующих этому DTD документов.

Ранее мы уже упоминали о различии между правильными и состоятельными документами, а также подробно рассмотрели вопрос, в чем же заключается *правильность* документа. Теперь наступил момент разобраться с тем, как определять структуру состоятельных документов. Следует учесть, что язык XML, как база данных, выстраиваемая формально по описанной схеме, также имеет формальный способ описания структуры документов.

Так как структуру XML нельзя объяснить линейно (то есть переходя последовательно от простого к сложному), мы, к сожалению, были вынуждены начать разговор на эту тему, предполагая, что читатели имеют некоторое представление о сути проблемы. На самом деле все подробные сведения, относящиеся к данному вопросу, будут изложены далее, в этой же самой главе. Ситуация непростая, но автору текущей части сборника остается только посоветовать читателям не сдаваться и продолжить работу, поскольку взаимосвязи различных аспектов языка XML, его целостный и многоликий образ, будет проясняться по мере нашего дальнейшего продвижения вперед.

Объявление XML

В начале второй главы мы узнали, что каждый XML-документ имеет пролог, содержащий объявление XML, за которым следует описание типа документа. Объявление XML управляет тем, каким образом анализатор будет интерпретировать документ.

Как вы помните, формат объявления XML тот же, что и формат команд приложений. Минимальное содержание объявления XML состоит из номера версии. Во время написания книги это был 1.0. Поэтому наши документы должны начинаться со следующей строки:

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

На самом деле объявление XML не обязательно, однако спецификация, утвержденная консорциумом W3C, предполагает, что его следует включать в документ для указания номера версии XML, использованной при создании документа. Это необходимо для того, чтобы выбрать соответствующий анализатор для обработки документа. Когда в документ включается объявление XML, первым обязательно должен идти номер версии, за ним (необязательно) название кодировки, за ними (необязательно) объявление автономности (см. ниже).

Кодировка набора символов в документе может быть при необходимости получена из кодировки самого документа. Каждый XML-анализатор должен поддерживать, по меньшей мере, 8-битовую кодировку Unicode, соответствующую символам ASCII. Анализатор может также поддерживать 16-битовую кодировку, которая содержит больше символов, чем позволяет ASCII. 8-битовая кодировка описывается следующим образом:

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

Другие возможные значения кодировок приведены в разделе 4.3.3 спецификации XML (смотрите «Приложение С»).

И наконец, атрибут `standalone` (автономный) описания документа указывает, имеет ли документ вне своих рамок какие-либо *описания разметки* (markup declarations) (об этом подробнее в следующих разделах), которые *могли бы повлиять* на величины, передаваемые приложению. Если значение `standalone` равно `'yes'`, это указывает, что документ самодостаточен. В этом случае не существует ни дополнительных описаний разметки, находящихся во внешних DTD, ни объявленных внутри данного документа внешних параметрических компонентов, содержащих такие описания (подробности тоже будут изложены позже). Ранее мы видели примеры объявления внешних компонентов в декларации `DOCTYPE`. При этом должна также, на случай если это окажется необходимым, существовать возможность обработки такого документа анализатором, не проверяющим его содержание на состоятельность.

Значение `'no'` для атрибута `standalone` указывает на то, что могут существовать описания разметки вне документа, то есть во внешнем подмножестве DTD или во внешних параметрических компонентах. Если, однако, описание разметки для внешнего компонента общего вида сделано во внутреннем подмножестве DTD, то ссылки на такой компонент внутри тела документа не мешают документу иметь

статус `standalone='yes'`.

Ограничения по состоятельности, которые накладываются спецификацией на описание `standalone`, описаны в спецификации (см. ниже).

Цитата

Атрибут standalone в документе должен иметь значение 'no', если во внешних описаниях разметки объявлены:

- *атрибуты со значениями по умолчанию, если элементы, которым эти атрибуты соответствуют, появляются в документе без значений, присвоенных данным атрибутам;*
- *компоненты (отличные от amp, lt, gt, apos, quot), если ссылки на эти компоненты появляются в документе;*
- *атрибуты, значения которых подлежат нормализации, в тех местах, где атрибут появляется в документе со значением, которое изменится в результате нормализации;*
- *типы элементов с элементным содержанием, если в каждой конкретной реализации этих типов непосредственно встречаются пробельные литеры (пробелы, символы табуляции, переводы строки, возвраты каретки).*

Если документ не является автономным, то полное объявление XML должно выглядеть так:

```
<?xml version="1.0" encoding="UTF-8" standalone="no">
```

Если значение `standalone` равно `'no'`, документ следует обрабатывать только верифицирующим анализатором, иначе данные, передаваемые приложению, будут неверными.

Описание типа документа

Как мы видели ранее, построение документа необходимо с чего-то начать. Такой исходной точкой служит корневой элемент, который для правильного документа определяется внутри описания типа документа.

Следует напомнить, что существует *определение типа документа* (Document Type Definition) и *описание типа документа* (Document Type Declaration). Их легко перепутать, так как описание типа документа включает в себя определение типа документа. Определение типа документа (DTD) охватывает все описания элементов документа, включая описания атрибутов, связанных с элементами. Определения типа документа также могут потребовать включения в свою структуру описания одного или более компонентов (либо внутренних по отношению к описанию типа документа, либо внешних). Описание типа документа представляет собой оператор вида `<!DOCTYPE somename [...]>`, и все описания DTD должны быть заключены между его квадратными скобками:

```
<!DOCTYPE arootname
[ <!ELEMENT arootname ...>

<!-- Другие описания элементов и связанных с ними атрибутов. -->
```

```
] >
<arootname>
   ... <!-- Здесь находится остальная часть документа. -->
</arootname>
```

Заметим, что имя `arootname` упоминается как в описании типа документа, так и в описании элемента `<!ELEMENT* ...>`. Необходимо включать в описание типа документа объявление элемента, соответствующего корневому элементу этого документа.

В теле описания типа документа можно зафиксировать все элементы и их атрибуты, а также любые компоненты и обозначения, определяющие его форму. Разрешается включать и такие описания, которые не нужны в данном экземпляре документа. При этом, правда, анализатору потребуется больше времени на прочтение описания типа документа, и он может затребовать дополнительные ресурсы при его обработке, однако включение подобных сведений не является ошибкой.

Определение типа документа

Определение типа документа состоит из описаний, находящихся внутри декларации `DOCTYPE` и включающих описания компонентов, элементов, атрибутов, а также любые другие описания, требующиеся для данного документа. Для удобства пользователя не обязательно включать все DTD в каждый документ из комплекта материалов одинаковой структуры – очень утомительно копировать одно и то же DTD в каждый документ, с которым мы работаем. Следовательно, возникает необходимость в способе, который позволил бы ссылаться на определение типа документа, хранящееся в отдельном файле. XML как раз и обеспечивает такой механизм. Теперь можно внутри каждого документа сослаться на внешний файл определения типа документа, используя следующий синтаксис:

```
<!DOCTYPE wrox SYSTEM "wrox-book.dtd">
```

Просто, не правда ли? Если оператор `DOCTYPE` содержит ключевое слово `SYSTEM`, за которым в качестве значения следует адрес URI, то процессор считывает определение типа документа из означенного URI и проверяет остальную часть документа на состоятельность в соответствии с этим определением. DTD, на которое ссылаются подобным образом, не обязательно должно быть составлено автором документа. При наличии разрешения можно использовать DTD, написанные другими авторами и подготовленные для широко распространенных форматов документов.

Совет *Посетите сайт http://www.schema.net, где можно найти примеры DTD для распространенных форматов XML-документов, находящиеся в свободном доступе.*

Как уже упоминалось в разделах, касавшихся правильных документов, декларация `DOCTYPE` имеет две части: внутреннее подмножество и внешнее подмножество. Описания можно располагать в обеих частях:

```
<!DOCTYPE full/structure SYSTEM "wrox-book.dtd"
[
```

```
<!ELEMENT para ( em | quote | blah )>
]>
```

Квадратные скобки `[ ]` содержат внутреннее подмножество DTD. Идентификатор `SYSTEM` обозначает внешнее подмножество. Если нам приходится использовать стандартное определение типа документа, можно, кроме того, вставлять имя общедоступного идентификатора (так же, как и в случае использования внешнего компонента – но об этом позже).

Для некоего стандартного набора материалов, подчиняющихся определенной схеме, предоставляемая XML возможность использования внешнего DTD означает, что достаточно написать одно-единственное DTD, на которое затем могут ссылаться все документы соответствующего перечня. Если все подобные документы соответствуют схеме, определенной во внешнем DTD, то каждый из них без исключения будет считаться не только правильным, но и состоятельным.

XML-анализатор сначала считывает внутреннее подмножество. Если во внешнем подмножестве встречаются какие-либо конфликтующие описания, они считываются, но игнорируются. В действительности анализатор использует первое из тех описаний, с которыми он встречается в описании типа документа. Это позволяет при необходимости заменять для конкретного документа некоторые определения, данные во внешнем DTD.

Описание элементов

Каждый документ состоит из одной или нескольких частей: например, в этой книге существуют главы и абзацы. В XML подобные части называются *элементами* (elements). Для каждого документа разрешено определить столько элементов, сколько потребуется. Элемент может состоять из других элементов или *символьных данных* (character data). Как мы увидим позже в разделе «Атрибуты типа ENTITY», некоторые элементы резервируют место для содержания другого типа (не XML) и, если его нет, остаются пустыми.

Главу из этой книги можно, например, определить следующим образом:

```
<!ELEMENT chapter (title, text) >
```

Это определение указывает, что каждый элемент `chapter` (глава) состоит из двух других элементов, называемых `title` (заголовок) и `text` (текст), которые необходимо ообозначить в их собственных описаниях. Каждый элемент оформляется одинаково: символ `<`, за которым сразу же следует слово `!ELEMENT`, затем имя нового элемента, содержание элемента и в конце символ `>`. В нашем примере тэг главы может быть использован следующим образом:

```
<chapter>
   <title>..</title>
   <text>..</text>
</chapter>
```

Существуют особые правила, указывающие, как правильно записываются *имена* (names) элементов. Некоторые символы, которые нам хотелось бы использовать, зарезервированы в XML для других целей, другие же пропущены для поддержки совместимости с SGML.

Уточнение *Для имен в XML разрешается использовать буквы, цифры и следующие знаки пунктуации: символ подчеркивания, дефис и точка. Имена не могут начинаться с цифры. Недопустимо использовать кавычки и знак вопроса, поскольку эти символы зарезервированы в языке XML для других целей.*

Понятие буквы в XML гораздо шире, чем можно себе представить. Если в вашем родном языке больше букв, чем в алфавите английского языка, считайте, вам повезло. До тех пор пока вы в соответствии со стандартом Unicode используете символы, которые считаются «буквами», эти знаки и в XML будет считаться буквами. Таким образом, имена в XML можно записывать на любом языке (если только набор его символов поддерживается стандартом Unicode).

В том случае, если внутри документа имеется правильное вхождение некоторого элемента, но тип самого элемента (и связанных с ним атрибутов) не описан, элемент, конечно, не может относиться к числу состоятельных. Только тот элемент может считаться состоятельным, для которого:

- существует описание типа элемента в DTD, имеющее то же имя, что и сам элемент;
- существуют описания в DTD всех атрибутов для этого типа элемента, а также типов значений атрибутов;
- тип данных в содержании элемента соответствует типу данных в схеме содержания, определенной в описании.

Правда, на описание элементов есть особое ограничение по состоятельности, которое заключается в том, что для одного и того же документа никакой *тип* элемента не может быть воспроизведен более одного раза. Это означает, что внутри одного и того же документа нельзя иметь более одного описания с одинаковыми именами типа элемента (включая любую часть DTD документа, которая физически находится вне документа). То есть нельзя включить оба следующих описания в одно и то же DTD, поскольку они относятся к элементам с одним и тем же именем `chapter`:

```
<!ELEMENT chapter (title, text) >
<!ELEMENT chapter (religious_order, number_of_nuns) >
```

Если бы возникла необходимость поместить оба этих элемента в наш документ, конфликта имен можно было бы избежать, назвав элементы CHAPTER и chapter, поскольку XML различает заглавные и строчные буквы. Тем не менее, следует помнить, что некоторые старые анализаторы перед обработкой заменяют буквы имен тэгов на заглавные, тем самым вызывая противоречие в употреблении имен. А в том случае, если вы захотите обработать этот же документ SGML-анализатором, имейте в виду, что по умолчанию SGML не различает заглавных и строчных букв.

Модели содержания элемента

Каждый элемент описывается *моделью содержания* (content model), находящейся внутри его описания и перечисляющей все дочерние элементы, которые мо-

гут в нем содержаться. Некоторые из дочерних элементов могут повторяться или быть необязательными. Модель содержания является частью описания элемента и (не считая двух исключений, которые будут рассмотрены позже) заключается в скобки. Модель описывает, какие элементы нижнего уровня составляют текущий элемент. При условии выполнения правил на употребление имен дочерних элементов не существует ограничений.

Разберем следующий пример:

```
<!ELEMENT sample   (date.taken, volume, substance) >
```

Как видите, элемент `sample` состоит из трех дочерних элементов: `date.taken`, `volume` и `substance` в заданном порядке. В документе они появятся таким образом:

```
<sample>
       <date.taken>..</date.taken>
       <volume>..</volume>
       <substance>..</substance>
</sample>
```

«Язык», применяемый в моделях содержания, тот же, что и в стандартных выражениях операционной системы Unix, используемых для поиска и замены в документах, просматриваемых под управлением этой системы. Если вы знакомы со стандартными выражениями Unix, вам будет легко читать наши модели содержания. При этом следует учитывать, что в моделях содержания находится не образец текста для поиска, как в стандартных выражениях, а исключительно элементы. Символы, употребляемые в моделях содержания XML, являются подмножеством символов, применяемых в стандартных выражениях. Например, скобки используются для группирования, знак вопроса – для указания необязательных элементов, звездочка и плюс указывают на повторение. Нам также встретится запятая, обозначающая последовательное расположение элементов; в последнем случае в регулярных выражениях будем считать, что элементы выражения расположены рядом.

В табл. 2.3 приведены все операторы модели содержания и их использование.

Таблица 2.3. Операторы модели содержания

Символ	Использование
, (запятая)	Строгая очередность
| (вертикальная черта)	Выбор
+ (знак плюс)	Повторение (минимум 1 раз)
* (звездочка)	Повторение
? (знак вопроса)	Необязательно
( ) (скобки)	Группирование

Рассмотрим пример модели содержания для элемента `test`:

```
<!ELEMENT test (sample+, conclusion) >
```

Теперь элемент `sample` является частью модели содержания элемента более высокого уровня `test`. Знак `+` указывает, что должен иметься, по крайней мере,

один элемент sample, хотя их может быть и больше. После последнего элемента sample должен стоять один элемент conclusion. В документе это будет представлено следующим образом:

```
<test>
       <sample>..</sample>
       <sample>..</sample>
       ..
       <sample>..</sample>
       <conclusion>..</conclusion>
</test>
```

Если бы мы изменили модель на (sample, conclusion)+, то за каждым элементом sample должен был бы следовать элемент conclusion, и внутри элемента <test> имелась бы, по крайней мере, одна пара sample-conclusion. Использование запятой в модели содержания означает, что дочерние элементы должны следовать друг за другом в заданном порядке. Поэтому для измененной модели мы бы получили в документе что-то вроде:

```
<test>
       <sample>..</sample>
       <conclusion>..</conclusion>
       ..
       <sample>..</sample>
       <conclusion>..</conclusion>
</test>
```

где все дочерние элементы, кроме первой пары sample-conclusion, необязательны.

Заменив запятую вертикальной чертой, получим модель «или... или». То есть (sample|conclusion) означает, что следует ожидать наличия либо одного элемента sample, либо одного элемента conclusion, но не обоих. Итак, в документе мы имеем запись вида:

```
<test>
       <sample>..</sample>
</test>
```

или запись вида:

```
<test>
       <conclusion>..</conclusion>
</test>
```

Следует соблюдать осторожность при использовании вертикальной черты, или *соединителя выбора* (selection connector). Оба элемента не могут быть введены одновременно; включается строго либо один, либо другой. Не путайте эту запись с типичным использованием символа вертикальной черты в языках программирования для обозначения операций «поразрядного ИЛИ» над данными в двоичном формате.

Постановка в конце выражения знака ?, то есть (sample|conclusion)?, сделало

бы включение любого из этих двух элементов необязательным. Таким образом, в добавление к двум предыдущим примерам, получаем:

```
<test>
</test>
```

Наоборот, при использовании символа обобщения `*` (звездочка), можно включать наши элементы, а можно их вообще игнорировать. Подобный результат отличается от того, что предлагает модель `(sample|conclusion)?`. Модель `(sample| conclusion)*` позволяет ввводить любое количество элементов `sample` или `conclusion`, а также совсем обходиться без них. Например, можно записать так:

```
<test>
       <sample>..</sample>
       <sample>..</sample>
       ..
       <sample>..</sample>
</test>
```

или использовать беспорядочную смесь обоих элементов в виде:

```
<test>
       <sample>..</sample>
       <conclusion>..</conclusion>
       <conclusion>..</conclusion>
       <sample>..</sample>
       ..
       <sample>..</sample>
       <conclusion>..</conclusion>
       <sample>..</sample>
</test>
```

или в виде:

```
<test>
       <conclusion>..</conclusion>
       <conclusion>..</conclusion>
       ..
       <conclusion>..</conclusion>
</test>
```

или в виде:

```
<test>
</test>
```

Внутри группы `(...)` соединительные символы (запятая или вертикальная черта) должны быть одинаковыми. Их нельзя перемешивать `(sample+, test|conclusion)`, не вводя новые скобки `(sample+, (test|conclusion))`. В отличие от требований математики, никакого определенного отношения старшинства соединительных символов здесь не подразумевается.

Когда анализатор, проверяющий XML-документ на состоятельность, просматривает материал, он использует модели содержания, чтобы убедиться, что каждый

элемент в документе вставлен на предназначенное ему место.

Табл. 2.4 демонстрирует синтаксис различных описаний содержания элемента.

Таблица 2.4. Синтаксис описаний содержания элемента

Модель содержания	Комментарии
x	Ожидается один элемент (x)
x | y	Один элемент (ожидается или x или y)
x, y	Ожидаются два элемента (x и y) в указанном порядке
x, y?	Разрешены два элемента (x ожидается, y по выбору)
(x, y, z)	Ожидаются три элемента x, за ним y, за ним z
x, (y | z)	Разрешены два элемента x ожидается, за ним или y или z
x, (y | z)*	Разрешены два элемента x ожидается, за ним в любом количестве y и z, или ни один из них
x, (y | z)+	По крайней мере два элемента ожидаются Сначала x, за ним один из y или z За этим может следовать любое количество y и z
x | (y, z)	Ожидается или один элемент x или y, а за ним z

Сложные группы могут быть построены заменой групп элементов в скобках на x, y и z.

Ниже приведен документ с внутренним DTD, похожим на реальное. В нем используются все сведения, изложенные ранее. Пожалуйста, обратите внимание на то, что многие тэги, например subA, myA, myB и т. д., не описаны, то есть, строго говоря, этот документ несостоятельный, однако он служит хорошим примером использования моделей содержания:

```
<?xml version ="1.0">
   <!DOCTYPE exampledoc [
   <!ELEMENT TopOne          (subA | subB)>
   <!ELEMENT subB            (myA, myB, myC)>
   <!ELEMENT Another         (First, Gorilla?)>
   <!ELEMENT Gorilla         (Y)+>
   <!ELEMENT almostLast      (giraffe, Zebra)*>
   ]>

<exampledoc>
   <TopOne>
       <subA> ... </subA>
   </TopOne>
   <TopOne>
       <subB>
           <myA> ... </myA>
           <myB> ... </myB>
           <myC> ... </myC>
       </subB>
   </TopOne>
   <Another>
       <First> ... </First>
   </Another>
```

```
    <Another>
        <First> ... </First>
        <Gorilla>
            <!-- Должен присутствовать хотя бы один Y.. -->
            <Y> ... </Y>
            ...
            <!-- ..все остальные необязательны. -->
            <Y> ... </Y>
        </Gorilla>
    </Another>
    <almostLast>
        <!-- Ноль или более пар: giraffe_а_потом_Zebra-->
        <giraffe> ... </giraffe>
        <Zebra> ... </Zebra>
        ...
        <giraffe> ... </giraffe>
        <Zebra> ... </Zebra>
    </almostLast>
</exampledoc>
```

Неоднозначные модели содержания

При использовании описанных выше операторов порой случайно возникают *неоднозначные* (ambiguous) модели содержания. Неоднозначность появляется в том случае, когда возможны две или более интерпретации одной и той же модели. Разработчики языка XML обычно используют очень строгое определение однозначности. В каждой точке они ожидают существование только одного подходящего элемента. Необходимо добавить, что хорошие верифицирующие анализаторы сразу, как только они сталкиваются с неоднозначными описаниями, должны выдавать ошибку.

Нетрудно заметить, что все рассмотренные в этой главе примеры можно назвать однозначными. Когда же один и тот же элемент обнаруживается в модели более одного раза, возможно появление неоднозначности. Рассмотрим следующий пример:

```
<!ELEMENT ambig ((date.taken, volume, substance) |
   (date.taken, stolen.article))>
```

Анализатор встречает в документе элемент `date.taken`. Возникает вопрос – должен ли за ним следовать элемент `volume` или элемент `stolen.article`? Вообще говоря, анализатор в момент обработки не знает, какую из двух возможных моделей содержания он должен использовать для элемента `<ambig>`. Если анализатор не способен проверить, следует ли за элементом `date.taken` элемент `volume` или элемент `stolen.article`, а затем вернуться назад и определить, нет ли какой-либо несовместимости, он не сможет обработать документ должным образом. Однако беда невелика – модель содержания всегда может быть изменена таким образом, что неоднозначность исчезнет:

```
<!ELEMENT ambig (date.taken, ((volume, substance) |
   stolen.article))>
```

В первой версии, когда встречался начальный тэг элемента `<ambig>`, было

неизвестно, следует ли ожидать после элемента `date.taken` элемент `volume` или элемент `stolen.article`. Поскольку речь шла об элементе `ambig`, выбор надо было сделать до того, как можно было увидеть элемент, следующий за элементом `date.taken`. Таким образом, модель содержания становилась неоднозначной. Во второй версии, когда виден начальный тэг элемента `ambig`, мы знаем, что за ним должен следовать элемент `date.taken`. Только после этого мы должны сделать выбор, но здесь уже ясно, какой именно элемент – `volume` или `stolen.article` – следует за элементом `date.taken`. Разница между двумя версиями заключается в том, что в первой анализатор при построении логического дерева элемента `<ambig>` до принятия решения обязан видеть логический путь на два уровня вперед. Во втором – анализатор должен в каждый момент работать пошагово. Большинство существующих сейчас анализаторов вряд ли имеют сложную встроенную часть, которая могла бы справиться с первой версией, и при попытке обработки таких документов скорей всего выдадут ошибку. Сразу заметим, что спецификация XML требует создания таких моделей содержания, при работе с которыми анализатор в каждый фиксированный момент времени мог бы продвигаться не более, чем на один шаг.

Бывает, что неоднозначность возникает при использовании индикаторов необязательного присутствия (то есть звездочки и знака вопроса – `*` и `?`).

```
<!ELEMENT ambig2   (date.taken?, (date.taken, volume, substance)) >
```

Обнаружив подобную модель содержания, анализатор не может решить, каким именно элементом является первый тэг `date.taken` – обязательным или нет. Эта модель может быть легко исправлена следующим образом:

```
<!ELEMENT ambig2   (date.taken, (date.taken?, volume, substance)) >
```

Символьные данные #PCDATA

Наконец мы добрались до элемента, сплошь состоящего из символов. Его модель содержания должна включать специальное имя элемента `#PCDATA`, при этом можно использовать имя `#PCDATA` в модели содержания само по себе:

```
<!ELEMENT pname (#PCDATA) >
```

Название `#PCDATA` напоминает о том, что в этом месте документа придется иметь дело с *разбираемыми символьными данными* (parsed character data). Символ `#` используется, чтобы имя `PCDATA` не было интерпретировано как имя элемента. Этот знак называется *индикатором зарезервированного имени* (reserved name indicator).

При этом, однако, не существует запрета создавать модели содержания вида `(#PCDATA|pcdata)`. Правда, пользователи, которым придется иметь дело с подобным, изобретенным вами определением типа документа скорей всего восторга не испытают.

Часто возникает необходимость построения моделей содержания элемента, которые включают данные `#PCDATA` наряду с другими дочерними элементами.

```
<!ELEMENT pname (#PCDATA | TweetyElem | SylvesterElem )*>
```

Эту работу следует выполнять предельно аккуратно, особенно когда существу-

ет вероятность, что документы будут разбираться не только XML-анализатором, но в каких-то случаях и SGML-анализатором. Эти последние в процессе обработки отбрасывают пробелы и некоторые другие символы форматирования внутри элементов, и вполне могут повредить концы записей. Если возникает необходимость использовать уже рассмотренное нами *смешанное содержание* (mixed content), следует продумать, как обращаться с концами записей, используя в различных элементах документа атрибут `xml:space`, отвечающий за сохранение пробельных литер (о нем поговорим позже).

Когда вы создаете описание смешанного содержания, `#PCDATA`, любые другие элементы группы должны быть разделены вертикальной чертой. Нельзя употреблять запятую для разделения членов в группах со смешанным содержанием.

Подразумевается, что любое содержание `#PCDATA` необязательно. Поэтому запись:

```
<pname></pname>
```

допустима, если описано, что элемент `pname` содержит данные `#PCDATA`.

Уточнение *Описания моделей смешанного содержания не должны использовать одно и то же имя дважды.*

Обратите внимени, что приведенная ниже модель содержания не может считаться правильной, поскольку дочерний элемент `evil_clone` встречается дважды:

```
<!ELEMENT  BadMixedRepeater  (#PCDATA | evil_clone |
   Dolly | sheep | evil_clone)*>
```

Модель содержания EMPTY

Иногда содержание элемента вообще не является текстовым материалом. Бывают случаи, когда необходимо применить какое-то имя элемента, чтобы зарезервировать место в структуре документа, а также дать указание, чтобы XML-анализатор не разбирал включенное в него содержание. Например, необходимо вставить графическое изображение. Все равно с точки зрения анализатора подобный элемент остается пустым. (В разделе, посвященном атрибутам, встретится пример его применения). Для этой цели можно использовать модель содержания `EMPTY` (пустое):

```
<!ELEMENT artwork  EMPTY>
```

Здесь нет скобок, и поэтому индикатор зарезервированного имени (символ `#`) в этом случае не требуется. Пустые элементы не нуждаются в присутствии завершающего конечного тэга, однако в конце единственного тэга они всегда должны завершаться косой чертой. Таким образом, элемент `artwork` в документе мог бы иметь следующий вид:

```
<artwork />
```

Модель содержания ANY

Работая с XML, можно встретить и такую необычную модель содержания, как `ANY` (произвольное); она используется редко и еще реже обсуждается в SGML/XML-сообществе. В кругу специалистов сложилось мнение, что при разработке определения типа документа такая модель как временная мера может оказаться полезной; в частности, если вы хотите сконструировать DTD по имеющемуся

XML-документу.

ANY позволяет включать в элемент любой тип содержания (даже #PCDATA), особенно, когда возникают ситуации, в которых допустимо появление любого типа содержания. Если что-то подобное имеет место, модель содержания принимает вид:

```
<!ELEMENT ElemName ANY >
```

И опять же не нужны ни скобки, ни индикатор зарезервированного имени.

Описание списка атрибутов

Как уже было сказано, элемент может иметь связанные с ним атрибуты. Их описания начинаются с команды <!ATTLIST... и содержат имя элемента, которому принадлежат атрибуты, имена атрибутов элемента, типы данных или допустимых значений содержания атрибутов, их значения по умолчанию. Приведем пример:

```
<!ELEMENT mycar EMPTY >
<!ATTLIST mycar
   is_blue         (yes|no)   "yes"
   brandname       ID         #REQUIRED
   description     CDATA      #IMPLIED
>
```

Каждый атрибут имеет три компонента: имя (например, is_blue), которое соответствует тем же правилам, что и имена элементов; тип передаваемой информации, а также способ обработки значений по умолчанию. (Подробнее о значениях ключевых слов #REQUIRED, #IMPLIED, #FIXED вы можете узнать в разделе «Значения атрибутов по умолчанию», включенном в эту главу.)

Элемент mycar используется внутри документа следующим способом:

```
<root>
   <mycar isblue="no"  brandname="68Studebaker" description="My second home" />
</root>
```

Элемент может иметь любое количество связанных с ним атрибутов. Вы устанете их вводить, прежде чем анализатор соберется выдать сообщение об ошибке.

С помощью DTD можно управлять интерпретацией значения атрибута. В правильных документах значение атрибута всегда представляется как содержащее символьные данные, так как не существует описаний атрибутов, которые могли бы нам сообщить другую информацию. Однако в случае состоятельных документов допустимы атрибуты с различными свойствами.

Для того чтобы атрибут был состоятельным, его тип должен быть объявлен в описании ATTLIST, и его значение должно соответствовать указанному типу данных.

Небольшое замечание перед тем, как перейти к рассмотрению различных типов атрибутов. Необходимо запомнить, что значения атрибутов, определенные в описании ATTLIST, не должны содержать символы «меньше чем», амперсанд, запятую, двойные или одинарные кавычки в литерной форме, поскольку во время разбора они будут интерпретированы как разметка, и в результате документ не может считаться правильным. На самом деле значения атрибутов в описании ATTLIST

в некоторых случаях могут содержать амперсанд, если он является частью ссылки на внутренний компонент, который был воспроизведен до описания ATTLIST. Учтите, что ни одно из приведенных ниже значений атрибутов не допустимо:

```
<!ATTLIST  elementName
   Bad_attrib1 #FIXED         " 10 < 15"
   Bad_attrib2 #FIXED         " Custard & French fries"
   Bad_attrib3 #FIXED         " Thus " spoke Zoomer"
   Bad_attrib4 #FIXED         " Hooray ' said Dan"     >
```

Однако следующие записи считаются разрешенными:

```
<!ENTITY   longdescription
   "yadayadayadayadayadayadayadayadayadayadayadayada......." >
<!ENTITY   lt  "&#38;#60;" >
<!ATTLIST  elementName
   okAttrib    #FIXED         "&longdescription; 10 &lt; 15"    >
```

(Подробная информация по данному вопросу приводится в разделе этой же главы, посвященном компонентам.)

Атрибут CDATA

Способ, с помощью которого анализатор обрабатывает значения атрибутов, зависит от того, как они описаны. В случае правильных документов предполагается, что значения атрибутов состоят только из символьных данных. Если необходимо сохранить ту же самую интерпретацию для состоятельного документа, следует описать атрибуты как CDATA:

```
<!ELEMENT with.attributes (#PCDATA)>
<!ATTLIST with.attributes
   sample  CDATA   #IMPLIED
>
```

Используя этот элемент в документе, атрибут можно задать в виде:

```
<with.attributes sample="Any old text provided that it isn't
   meant to have markup in it">
```

Когда атрибуту типа CDATA передается какое-либо значение, анализатор практически ничего с ним не делает. Предполагается, что приложение, использующее анализатор, само способно решить, как следует обрабатывать это значение.

Атрибуты ID и IDREF

Как только программист приступает к работе с состоятельными документами, перед ним сразу открывается широкий выбор операций, которые можно осуществлять над значениями атрибутов. Возможно, кое-кто из читателей уже привык использовать внутренние ссылки на Web-страницах, то есть щелкать мышью по «горячей» точке и переходить на другую часть страницы. Такой же механизм определения ссылок внутри документа существует и в языке XML. (Если быть точным, в главной спецификации XML 1.0 не существует механизма для определения связей между документами, однако в пятой главе, посвященной созданию ссылок в пространстве XML, будет рассмотрен дополнительный перечень требований,

которые как раз и обеспечивает подобный механизм.)

Прежде чем создать ссылку на некоторое место внутри документа, его надо как-то обозначить. Это можно сделать с помощью атрибута `ID`.

```
<!ELEMENT  Intralink   EMPTY>
<!ATTLIST  Intralink
   arrive_at   ID  #REQUIRED
>
```

Каждый элемент может иметь только один атрибут `ID`.

С помощью ключевого слова `#REQUIRED` мы устанавливаем, что каждый раз, когда встречается элемент `Intralink`, в нем обязан присутствовать сопровождающий его атрибут `arrive_at`.

Значение по умолчанию атрибутов типа `ID` должно быть или `#REQUIRED`, или `#IMPLIED`.

Элемент `Intralink` должен появиться в документе в таком виде:

```
<Intralink arrive_at="ReferToThisName"/>
```

Для этого элемента не существует отдельного конечного тэга, поскольку мы определили `Intralink` как пустой элемент.

В каждом случае элемент `Intralink` может иметь только один атрибут `ID` (который мы назвали `arrive_at`), и каждый экземпляр этого атрибута должен иметь уникальное значение.

Уточнение *Значение атрибута ID должно быть именем, которое только один раз может употребляться во всем документе.*

Таким образом, в приведенном выше примере имя `'ReferToThisName'` может быть использовано в качестве значения только для одного-единственного и никакого другого элемента внутри документа. То есть следующая запись недопустима:

```
<Intralink arrive_at="ReferToThisName"/>
<Intralink arrive_at="ReferToThisName"/>
```

В то же время вполне разрешено следующее выражение:

```
<Intralink arrive_at="ReferToUniqueOne"/>
<Intralink arrive_at="ReferToUniqueTwo"/>
<Intralink arrive_at="ReferToUniqueThree"/>
<Intralink arrive_at="ReferToUniqueFour"/>
```

Как только будут определены элементы, на которые мы намерены сослаться, следует использовать атрибуты `IDREF` для ссылок на них из других элементов:

```
<!ELEMENT refIntralink (#PCDATA)>
<!ATTLIST  refIntralink
   link_to     IDREF   #IMPLIED
>
```

Обратите внимание, что нам не нужно всякий раз описывать `IDREF` в качестве обязательного `(#REQUIRED)` атрибута, как это было сделано в случае атрибута `ID`. Несмотря на то, что мы определили атрибут `link_to` для элемента `refIntralink`,

нам совсем не требуется использовать его в каждом случае, когда в документе встречается тэг refIntralink. С помощью ключевого слова #IMPLIED можно сделать его использование необязательным.

Если требуется применить тэг, который позволяет создать большое число ссылок на различные места документа, можно использовать атрибут типа IDREFS.

```
<!ELEMENT Manyrefs                    (#PCDATA)>
<!ATTLIST Manyrefs
   link_to_many    IDREFS             #IMPLIED
>
```

Значения атрибута ID, на которые ссылается атрибут IDREFS, отделяются друг от друга одиночным символом пробела.

При использовании атрибутов получаем:

```
<Intralink arrive_at ="Figure_1"/>
<Intralink arrive_at ="Figure_2"/>
<Intralink arrive_at ="Figure_3"/>
<refIntralink link_to="Figure_1">This element refers to Figure 1
   </refIntralink>
<Manyrefs link_to_many="Figure_1 Figure_2 Figure_3">
   This tag refers to all three figures.
   </Manyrefs>
```

Для того чтобы быть состоятельным, значение атрибута IDREF должно совпадать с именем значения какого-либо атрибута ID в некотором элементе внутри документа. Имена, входящие в значение IDREFS должны подчиняться тому же правилу.

Атрибут ENTITY

Модели содержания являются полезным механизмом для описания структуры документов, но они не способны управлять типом данных, передаваемых как #PCDATA. Этот недостаток обычно проявляется при работе с двоичными данными. Проблему можно решить, включив для соответствующих элементов *атрибут ENTITY*.

Обычные компоненты, на которые ссылаются в документе, используя синтаксис &entity;, могут содержать только текст (разбираемые символьные данные или разметку XML). Если мы хотим включить в документы другие типы данных (например, в двоичном формате, такие как файлы с расширением .zip или файлы Java с расширением .class), то необходимо использовать атрибуты ENTITY (компонент).

Можно «поименовать» один компонент, обозначив атрибут типа ENTITY в описании ATTLIST элемента, к которому относится данный атрибут. Можно описать атрибут и так, чтобы он воспринимал несколько компонентов. Для этого следует использовать тип ENTITIES, который работает точно так же как тип IDREFS в предыдущем разделе. Приведем пример.

```
<!ELEMENT illustration (#PCDATA) >
<!ATTLIST illustration
   external.artwork    ENTITY         #IMPLIED
>
<!ENTITY    sune.logo   SYSTEM "Macintosh HD:XML Book:Chapter 2:entity
```

```
    example"    NDATA          PICT
>
<!ENTITY    WROX.logo    SYSTEM
    "http://www.wrox.com/images/smallogo.gif" NDATA     gif
>
<illustration external.artwork="sune.logo" ></illustration>
<illustration external.artwork="WROX.logo"></illustration>
```

Заметим также, что описания для двух внешних компонентов `sune.logo` и `WROX.logo` имеют тип данных `NDATA`. Эти компоненты следовало бы описать ранее, используя определение `<!NOTATION...>` (оно рассматривается в следующем разделе).

Компоненты, обозначенные `NDATA`, являются *неразбираемыми* (unparsed) внешними компонентами. Это означает, что данные, содержащиеся внутри этих компонентов, могут быть, а могут и не быть текстом; в первом случае текст может быть, а может и не быть кодом XML. Как бы то ни было, анализатор пропускает подробности содержания таких компонентов, поэтому не существует ограничений на тип данных, которые содержат неразбираемые компоненты. В примере, приведенном выше, неразбираемые внешние компоненты представляют собой файлы изображений, но неразбираемые компоненты могут быть и любого другого формата. Однако...

Уточнение *Единственный способ, с помощью которого можно включить неразбираемый компонент в документ – это использовать атрибут типа ENTITY для некоторого элемента. Такие атрибуты допускают в качестве значений только неразбираемые компоненты.*

Иначе говоря, для неразбираемого компонента недопустима отдельно стоящая ссылка с обычным синтаксисом `&some_entity;`. Имея дело с неразбираемыми компонентами, всегда нужно следовать образцу, приведенному в указанном выше примере.

Повторим еще раз, для того чтобы атрибут был состоятельным, его значение должно совпадать с именем неразбираемого (внешнего) компонента, описанного в DTD.

Атрибут NOTATION

В процессе создания документа бывают случаи, когда программисту необходимо, чтобы при определенных условиях имели бы место определенные последствия. Например, может понадобиться, чтобы при упоминании некоторых форматов файлов внутри конкретного документа обрабатывающее его приложение производило те или иные действия или вызывало другое приложение для обработки этих файлов.

Этого можно добиться, используя совместно описание `NOTATION` и атрибуты `NOTATION` (обозначение). Сначала введем описание `NOTATION`.

```
<!NOTATION PICTSYSTEM  "PikEdit1.exe">
<!NOTATION jpg SYSTEM  "file://myserver/myfolder/JPEGEdit.exe">
<!NOTATION gif PUBLIC  "http://www.server.com/somedir/GifEditor.exe">
```

Затем можно использовать эти типы файлов в модели содержания атрибута:

```
<!ATTLIST clipart
   file.format NOTATION       ( jpg | gif )
>
```

Заметим, что нам совсем не обязательно включать все ранее определенные в описании NOTATION значения в модель содержания атрибута.

В результате использования описания NOTATION для определения типа данных, любое подмножество его значений можно впоследствии применить в моделях содержания атрибутов. Обычно данные для атрибутов NOTATION внешние, их формат не совпадает с форматом XML.

Если присоединить к атрибутам NOTATION атрибуты типа ENTITY, можно быть уверенным, что никакая из существенных деталей не будет забыта и все интерпретируется должным образом. В самом деле, как уже было показано ранее, единственный способ, с помощью которого можно включить неразбираемые данные в документ – это использовать атрибуты типа ENTITY.

```
<!ELEMENT graphic  EMPTY >
<!NOTATION gif PUBLIC  "_//Wrox//NOTATION WroxGifs format//EN">
<!ATTLIST graphic
   file.name   ENTITY  #REQUIRED
   file.format NOTATION       gif
>

<!ENTITY   LOGO.entity SYSTEM "C:\IMAGES\LOGO.gif"      NDATA   gif>

<graphic   file.name="LOGO.entity" file.format="gif" />
```

Обратите внимание, что LOGO.entity является внешним компонентом по отношению к документу, а его тип данных описывается как 'NDATA gif'. Использование ключевого слова NDATA указывает на то, что компонент имеет тип данных, который был ранее описан в NOTATION, и является неразбираемым.

Для состоятельности все имена обозначений предварительно должны быть определены с использованием описания NOTATION, а затем значение атрибута типа NOTATION должно совпасть с одним из имен обозначений, объявленных для элемента в описании ATTLIST.

Атрибут NMTOKEN

Часто случается, что пользователи испытывают необходимость в атрибутах, значением которых является имя. Такие имена могут быть использованы для названия языка или кода, а также защиты документа. В XML мы можем описать атрибут как NMTOKEN (ярлык). Значения таких атрибутов должны соответствовать тем же правилам, что и другие имена в XML, например элементов и атрибутов. Правда, подобные ярлыки не должны начинаться с символа подчеркивания или двоеточия.

```
<!ELEMENT  para    (#PCDATA | ... )*>
<!ATTLIST  para
   security    NMTOKEN #IMPLIED
```

```
>
...
<para>This paragraph has whatever the application's default security level
is.</para>
<para security="confidential">Whereas this paragraph is assumed to be confidential
because we used a NMTOKEN attribute.</para>
```

Далее атрибут `NMTOKEN` может быть использован приложением, которое, например, выбирает, кому именно можно демонстрировать определенные разделы информации внутри документа, а кому нет.

Значениями `NMTOKEN` следует пользоваться аккуратно, поскольку проверяется не их орфография, а только соответствие правилам записи имен в XML.

Иногда бывает необходимо в качестве значений атрибута включить в запись несколько имен. Если попытаться произвести эту операцию с помощью атрибута `NMTOKEN`, будет выдано сообщение об ошибке. В таких случаях следует использовать атрибут `NMTOKENS`, поскольку именно он позволяет включить более чем одно значение. Однако атрибут `NMTOKENS` можно использовать и тогда, когда требуется только одно значение. Если же значений несколько, они должны быть разделены одиночными пробелами. Разница между атрибутами `NMTOKEN` и `NMTOKENS` состоит в количестве значений, которые могут быть включены в начальный тэг.

```
<!ATTLIST  para
   secure.users    NMTOKENS   #IMPLIED
>
<para security="confidential" secure.users="JONSOND dj BOUMPHREYF fb">...
```

Вследствие изменения спецификации языка XML, обеспечившего различение заглавных и строчных букв, информация о том, какой именно является буква, сохраняется при передаче ее приложению. В примере, приведенном выше, когда были использованы имена входа в систему, это может оказаться существенным моментом. В отличие от случаев несоответствия имен элементов и компонентов, о которых анализатор сообщает сразу, в режиме значений `NMTOKEN` последствия подобного разнобоя зависят от способности приложения (и его автора) справиться с ними.

Атрибуты с перечисляемыми значениями

Атрибуты типа `NMTOKEN` можно использовать в любых целях. До тех пор пока его значение распознается как имя, оно допускается анализатором XML. Однако случается, что подобное решение может не устраивать программиста. Код приложения, вызывающего анализатор, может потребовать, чтобы имена в значениях записывались определенным образом. Приложение может быть написано так, чтобы воспринимать значение имени `"millennium"`, но, к несчастью, некоторые люди все равно напишут его неправильно: `"millenium"`. Подобные ошибки можно предупредить, перечислив все допустимые значения.

```
<!ATTLIST  para
   epoch.name  (millennium)  #IMPLIED
>
```

Если вы попытаетесь использовать для подобных целей значение `<para epoch.name="millenium">..`, то анализатор сообщит об ошибке точно так же, как

он сделал бы это и в том случае, если бы был использован вариант <para epoch.name="LotusYr1900">.

Когда необходимо задействовать несколько величин, можно перечислить их все и отделить друг от друга с помощью вертикальной черты. В языке SGML допустимо применять любой из соединителей (SEQUENCE, AND и OR), однако преимущество использования вертикальной черты в XML состоит в напоминании, что необходимо выбрать одно из нескольких допустимых значений. Если применяется атрибут choice, то все, что мы должны сделать – это перечислить допустимые значения, одно из которых должно быть использовано:

```
<!ATTLIST  para
   choices (this | that | other) #REQUIRED
>
```

Для того чтобы атрибут choice был состоятельным, значения, присваиваемые атрибуту, когда он встречается в начальном тэге элемента para, должны совпадать с одним значением из группы, описанной в ATTLIST.

Так же как и в случае со значениями атрибутов типа NMTOKEN, следует обратить особое внимание на употребление в перечисляемых значениях заглавных и строчных букв. Анализатор сообщит об ошибке, если хотя бы в одном из значений произошел сбой. Если необходимо употребить различные комбинации заглавных и строчных букв для одного и того же значения, необходимо перечислить их все в описании атрибута, а также в коде соответствующего приложения.

```
<!-- Все эти значения в анализаторах XML рассматриваются как различные. -->
<!ATTLIST  para
   case.choices    (this | This | tHis)       #REQUIRED
>
```

Следует быть осторожным с описаниями списка атрибутов, содержащими перечисляемые значения, если они впоследствии должны быть использованы анализатором SGML. До тех пор, пока ожидаемая техническая поправка еще не утверждена, все перечисляемые значения атрибута в SGML должны быть уникальными для данного элемента. Это условие касается совместимости XML- и SGML-анализаторов. В настоящее время подобное ограничение для SGML представляется не слишком удачным, но все-таки еще раз подчеркнем – до той поры, пока не будут введены в действие поправки в стандарте, следует соблюдать особую осторожность при описании атрибутов. Если предполагается продублировать документы как документы SGML, обратите внимание на пример, приведенный ниже:

```
<!ATTLIST  pars
   lockable    (yes | no)
   printable   (yes | no)
>
```

Это описание допустимо при обработке анализаторами XML, но существующие анализаторы SGML примут его за ошибку. Пользователи XML могут решить, что значения атрибутов lockable и printable можно считать уникальными (например, lockable-yes и printable-yes), однако в SGML значения должны быть глобально

уникальными, и SGML-анализатор примет два вхождения значения yes за конфликт имен, то есть за ошибку.

Значения атрибутов по умолчанию

Где-то в глубинах человеческого сознания непременно таится мысль о том, как бы попроще ввести значение по умолчанию и как бы пропустить атрибуты, неприменимые к текущему элементу. XML позволяет в некоторой степени ответить на этот зов души программиста. Вы наверняка заметили, что каждое описание атрибута в этой главе включало в себя слова #REQUIRED или #IMPLIED или #FIXED, но до сих пор мы не обращали внимания на то, что же они все-таки означают.

Атрибуты, отмеченные флагами #REQUIRED или #IMPLIED

Флаг #REQUIRED (обязательное) принуждает атрибут появляться в каждом начальном тэге данного элемента. Другими словами, для любого атрибута, помеченного таким флагом, не существует значения по умолчанию. В то же время, флаг #IMPLIED (неявное) указывает на то, что приложение может предоставить значение по умолчанию, если в конкретном документе оно не задано явным образом. При этом может случиться, что при работе с флагом #IMPLIED различные приложения при обработке одного и того же документа могут предоставлять разные значения по умолчанию, а это безусловно недопустимо. Несмотря на то, что подобный метод много лет применялся в SGML, к нему до сих пор относятся как к чужеродному телу, так как смысл его идет вразрез с установкой, что все должно быть сказано в разметке. Однако если создатель документа достаточно компетентен, он все-таки отметит флагом #REQUIRED те атрибуты, для которых присутствие предоставляемых значений имеет решающее значение. В то же время атрибуты, для которых значения, предоставляемые пользователем, не существенны (то есть те, для которых использование любого значения по умолчанию окажет минимальное или никакого влияния на приложение) будут отмечены флагом #IMPLIED.

Атрибуты, отмеченные флагом #FIXED

Любой атрибут, отмеченный флагом #FIXED (фиксированное), при использовании в данном элементе может иметь только одно определенное значение. В примере, приведенном ниже, атрибут spelling, если он появляется в элементе word, может иметь только фиксированное (#FIXED) значение: millennium.

```
<!ELEMENT  word    (#PCDATA>
<!ATTLIST  word
   spelling    CDATA   #FIXED "millennium"
>
<word spelling="millennium"></word>
```

Когда элемент word появляется в документе, он не обязательно должен содержать атрибут spelling в явном виде, поскольку описание устанавливает, что в отсутствие реального атрибута для элемента подразумевается его значение по умолчанию 'millennium'. Таким образом, если в документе появляется элемент:

```
<word> A little bit of text </word>
```

анализатор подразумевает под ним элемент:

```
<word spelling="millennium"> A little bit of text </word>
```

Значения по умолчанию для атрибутов с перечисляемыми значениями

Предположим, что вам потребовался атрибут, для которого допустимы два или три значения, а также значение по умолчанию, когда он не появляется в элементе. Дело в том, что многие, например, несмотря на все старания, пишут слово `'millennium'` неправильно. В этом случае возникает следующее описание:

```
<!ELEMENT  word    (#PCDATA)>
<!ATTLIST  word
   spelling    ( millenium | milenium | millennium )
       "millennium"
>
<word spelling="millennium"></word>
```

Это описание устанавливает, что любой из трех вариантов записи слова `'millennium'` является допустимым, а в отсутствие значения для атрибута анализатор подставит значение `'millennium'`. Можно в явном виде описать в группе сколько угодно подобных допустимых величин. Например:

```
<!ATTLIST  counting
   numbers ( one | two | three | four | and | so | on | ...)
       "ten"
>
```

В заключение отметим, для того, чтобы быть состоятельными, описанные для атрибута значения по умолчанию должны иметь тип данных, соответствующий этому атрибуту.

В табл. 2.5 приведены все возможные типы данных для значения атрибута.

Таблица 2.5. Возможные типы данных для значений атрибута

Тип атрибута	Содержание значения атрибута
CDATA	Символьные данные
ID	Имя, которое является уникальным среди всех других значений атрибута ID
IDREF	Имя, которое определяется другим атрибутом ID
IDREFS	Последовательность имен, которые определяются атрибутами ID. Имена разтделяются пропусками
ENTITY	Имя уже существующего внешнего компонента. Предполагается, что этот компонент содержит данные в двоичном формате
ENTITIES	Последовательность имен уже существующих внешних компонентов. Имена разтделяются пропусками. Предполагается, что компоненты содержат данные в двоичном формате
NMTOKEN	Имя, которое не может начинаться с подчеркивания или двоеточия
NMTOKENS	Последовательность значений NMTOKEN, разделенных пробельными литерами
NOTATION	Последовательность значений NMTOKEN, разделенных пробельными литерами, которым были присвоены имена в описаниях NOTATION
С перечисляемыми значениями	Последовательность значений NMTOKEN, которые были в явном виде перечислены в описании атрибута

Множественные описания атрибутов

Иногда возникает необходимость описывать атрибут в несколько этапов, например, при расширении атрибутов для элемента, определенного во внешнем подмножестве описания типа документа. Когда подобные множественные описания попадаются анализатору XML, тот рассматривает только первое и игнорирует остальные. Можно перекрыть любой атрибут, описав его заново, поскольку внутреннее подмножество считывается первым.

```
<!ELEMENT  over.ride.sample  (#PCDATA) >
<!ATTLIST  over.ride.sample
   attr.to.over.ride   CDATA  #IMPLIED
>
<!-- Анализатор проигнорирует следующее описание. -->
<!ATTLIST  over.ride.sample
   attr.to.over.ride   NOTATION       (gif)
       #FIXED "gif"
>
```

Когда все описания соединяются воедино, и определение типа документа начинает наконец выстраиваться, можно использовать порядок следования частей внутри декларации `DOCTYPE` для перекрытия определений атрибутов. Еще раз обращаем ваше внимание – именно внутреннее подмножество *считывается* первым, несмотря на то, что внешнее подмножество задается раньше.

```
<!DOCTYPE  book    PUBLIC    "-//BigBook Press//DTD Easyreader Series//EN"     [
   <!-- Это внутреннее подмножество, в котором можно ввести описания,
       перекрывающие все остальные описания. -->
   <!ATTLIST   artwork
   format  NOTATION    (TeX | PostScript |
       Private)    #IMPLIED
   >
]>
```

В этом примере предполагается, что элемент `artwork` имеет во внешнем описании связанный с ним атрибут `format`, который не включает обозначения `Private`.

Нормализация значений атрибутов

По умолчанию анализатор XML не передает исходные значения атрибутов в вызывающее приложение. Вместо этого, для экономии памяти и предотвращения неправильной интерпретации, он передает *нормализованную* (normalized) версию. Анализатор заменяет все *концы записей* (record ends) – обычно возврат каретки и переход на новую строку или только переход на новую строку – пробелами, после чего все последовательности пробелов сокращаются до одного. При использовании старого анализатора XML, который еще не различает заглавных и строчных букв, следует иметь в виду, что он преобразует все буквы в значениях атрибутов `NMTOKEN` в заглавные.

Зарезервированный атрибут xml:space

При обработке текстов типа поэтических произведений может понадобиться дополнительный атрибут, управляющий вставкой знаков конца строки; обычно для этого употребляют символ `'/'`, обозначающий переход на новую строку.

Однако в соответствии с соглашением, существует *зарезервированное имя атрибута* (`xml:space`), который можно связать с любым элементом, в котором имеют значение пробельные литеры (пробелы, табуляции, переводы строки, возвраты каретки). При включении этого атрибута со значением `'preserve'` в тэг элемента анализатор должен сохранить пробельные литеры и передать их в вызывающее приложение для обработки.

```
<!ELEMENT  poetry      (#PCDATA | %emphasis;)*>
<!ATTLIST  poetry
   xml:space   (default | preserve) 'preserve'
>
```

Для сохранения или упразднения пробельных литер корневой элемент сам по себе не имеет никакого неявного значения по умолчанию. Поэтому, до тех пор пока для корневого элемента не будет описан этот атрибут и не будет задано значение по умолчанию, вызывающее приложение будет разбирать корневой элемент (а следовательно, и все те части документа, для которых обработка пробельных литер не была задана в явном виде) в соответствии со значениями по умолчанию, принятыми в этом приложении.

Описание компонентов

Теперь пора поговорить о том, в каком виде наши документы сохраняются в файлах. При этом следует учесть, что XML использует более обобщенную концепцию компонентов, о которой мы уже вкратце упомянули при обсуждении атрибутов.

Как и препроцессор языка C или C++, механизм компонентов позволяет подключать файлы и определять макросы. Как мы позже увидим, существуют некоторые ограничения на то, где именно внутри документа можно описывать определенные типы компонентов, а также ссылаться на них. Еще одно условие – компоненты всегда обязаны быть завершенными. Так, например, если компонент содержит начальный тэг, он должен содержать и соответствующий ему конечный тэг; если компонент содержит начало описания, конец описания должен находиться в нем же.

Различается два типа компонентов, которые мы можем описывать. Первые называются *параметрическими компонентами* (parameter entities), их применяют только внутри описания типа документа. Другие именуются *обычными компонентами* (general entities). Ранее уже было сказано, что они делятся на внешние и внутренние. Они объявляются в описании типа документа, а используются только в его содержании.

Прежде всего приведем общее правило, касающееся состоятельности компонентов.

Уточнение *Состоятельный документ требует, чтобы имя, используемое в ссылке на компонент (&entityname; или %entityname;) внутри документа, совпадало с именем, заданным в описании компонента в DTD.*

То есть для каждой ссылки на компонент, которая встречается в документе, где-то в DTD обязательно должно существовать описание его типа.

Параметрические компоненты

Параметрические компоненты используются только внутри DTD. Предположим, имеется длинный список параметров, который должен входить в модели содержания нескольких элементов. Используя описание параметрического компонента в следующем виде:

```
<!ENTITY % lotsoftags "V1|V2|V3|V4|V5|X1|X2|X3|X4|X5|X6" >
```

можно сократить объем описаний других элементов или атрибутов. Знак процента (`%`) означает, что компонент является параметрическим. Например, воспользовавшись параметрическим компонентом, определенным выше, можно создать следующие описания:

```
<!ELEMENT  BigElem (value1, value2, (%lotsoftags;|#PCDATA)*)>
<!ELEMENT  SmallElem   (goat|sheep|wolf|bear|%lotsoftags;)?>
```

Уточнение *Указанным способом можно, однако, ссылаться на параметрические компоненты только во внешнем подмножестве DTD. Во внутреннем подмножестве можно вставлять ссылки на параметрические компоненты только между описаниями.*

Таким образом, в примере, приведенном выше, показаны описания, которые могли бы находиться во внешнем подмножестве. Обратите внимание, что во внутреннем подмножестве допустимы только такие варианты:

```
<!ELEMENT  BigElem (value1, value2, (#PCDATA))*>
%xternParamEnt;
<!ELEMENT  SmallElem   (goat|sheep|wolf|bear)?>
%anotherXternParamEnt;
```

Уточнение *Описания параметрических компонентов всегда должны предшествовать любым ссылкам на них в DTD.*

Поскольку внутреннее подмножество DTD документа считывается первым, в нем нельзя ссылаться на параметрические компоненты, описанные во внешнем подмножестве; анализатор тут же выдаст сообщение об ошибке. Однако внутри внешнего подмножества можно ссылаться на компоненты, описанные во внутреннем подмножестве.

В состоятельных документах также существует ограничение на использование ссылок на параметрические компоненты внутри моделей содержания элементов, аналогичных рассмотренным в предшествующем примере. Текст замены параметрического компонента, на который существует ссылка в группе, заключенной в скобки, не может содержать текстовую строку, которая привела бы к тому, что описание элемента перестало бы быть правильным после вставки текста. Таким образом, если текст замены параметрического компонента содержит открываю-

щую скобку, он также должен содержать и закрывающую скобку. Настоятельно рекомендуем, чтобы текст замены не был пустым, и чтобы первые или последние символы в строке не были запятой или вертикальной чертой.

Данные, которые заменяют собой ссылку на компонент, не могут быть заданы в DTD тем же способом, как в приведенном выше примере. Такие данные могут находиться во внешнем файле. Описание подобного компонента имеет вид:

```
<!ENTITY % ExtEntities SYSTEM
   "http://www.someserver.com/somedir/extentity.txt " >
%ExtEntities;
```

Считается хорошим тоном сразу же после описания внешних компонентов сослаться на них во внутреннем подмножестве состоятельного документа. Пока ссылка на такие компоненты не указана, данные или описания, которые в них содержатся, не составляют часть DTD.

Внешние параметрические компоненты, на которые ссылаются во внутреннем подмножестве, должны содержать такой текст, который будет правильным и после того, как заменит ссылку на компонент в DTD. Иначе говоря, разметка в тексте замены должна быть составлена из законченных описаний.

В этом случае в спецификации сработает указанное ниже ограничение для состоятельных документов.

Уточнение *В текст замены параметрического компонента должны быть правильно вложены описания разметки.*

Это правило применяется к параметрическим компонентам, как во внутреннем, так и во внешнем подмножестве DTD.

Обычные компоненты

В противоположность параметрическим компонентам, *обычные компоненты* (general entities) применяются в тех случаях, когда необходимо упростить сами документы, а не определения их типов. Мы уже встречались с формой, которую принимают описания обычных компонентов:

```
<!ENTITY   artworkmode1              "Jean Jacques Rousseau">
<!ENTITY   bigtext SYSTEM            "http://www.server.com/dir/bigfile.txt" >
```

Для состоятельного документа существуют дополнительные правила, которые нужно соблюдать при работе с обычными компонентами.

Внутренние компоненты и значения атрибутов по умолчанию

Вы наверняка запомнили приводимую ранее табл. 2.2, указывающую, как процессор обрабатывает компоненты в зависимости от их содержания. Надеемся, вы также помните о разрешении ссылаться на внутренние компоненты в значениях атрибутов. При проверке на состоятельность это, в свою очередь, обуславливает, что ссылки на компоненты внутреннего типа могут также встречаться в значении по умолчанию, определенном в описании `ATTLIST` атрибута. Если последнее имеет место, тогда описание компонента должно предшествовать описанию атрибута. Итак, следующая запись считается правильной:

```
<!ELEMENT  secret_info EMPTY >
<!ENTITY   access_pass_no             "094850157132857-13097310987309457147" >
<!ATTLIST  secret_info
   password    CDATA   #FIXED "Access Permitted: &access_pass_no;" >
```

В свою очередь, описания, приведенные ниже, неверны, потому что верифицирующий анализатор не будет знать, каким образом интерпретировать ссылку `&access_pass_no;`.

```
<!ELEMENT  secret_info EMPTY >
<!ATTLIST  secret_info
   password    CDATA   #FIXED "Access Permitted: &access_pass_no;" >
<!ENTITY   access_pass_no             "094850157132857-13097310987309457147" >
```

Обязательные компоненты

XML дает возможность, использовать в правильном документе компоненты, соответствующие символам «меньше чем» (<), «больше чем» (>), «амперсанд» (&), а также двойные и одинарные кавычки, не описывая их. Однако в состоятельных документах эти компоненты должны быть зафиксированы в DTD следующим способом:

```
<!ENTITY   lt          "&#38;#60;" >
<!ENTITY   gt          "&#40;" >
<!ENTITY   amp         "&#38;#38;" >
<!ENTITY   apos        "&#39;" >
<!ENTITY   quot        "&#34;" >
```

Напоминаем, что следует задавать отложенный разбор компонентов `lt` и `amp`, поскольку в соответствии со спецификацией XML, утвержденной консорциумом W3C, описания этих компонентов должны быть правильными.

Внешние компоненты

В конце концов, может случиться, что количество компонентов, находящихся в работе, станет слишком велико, чтобы их можно было с помощью команды вырезки-вставки переносить из одного DTD-файла в другой. Иногда бывают также компоненты с очень длинными текстами вставок. Если пользователь предполагает, что компоненты должны быть общими для многих документов, то копировать их в определение типа документа допустимо, но если компоненты должны быть общими для многих определений типов документов, тогда копирование становится нерациональным. Можно сэкономить время и усилия, если попытаться сохранить текст замены компонентов отдельно от документов и любых внешних DTD. В XML эти компоненты, известные как *внешние компоненты* (external entities), практически идентичны неразбираемым компонентам, которые используются с атрибутами типа `ENTITY`. Разница между ними состоит в том, что первые могут содержать только текст или XML-данные.

Внешний компонент может быть включен в состав документа (путем создания ссылки на него) точно таким же способом, каким создаются ссылки на внутренний компонент.

```
<!ENTITY   copyright.statement        SYSTEM "Macintosh HD:XML Book:Copyright
```

```
Text">
...
&copyright.statement;
```

Когда анализатор встречает ссылку на внешний компонент, он просматривает определение этого компонента и начинает искать место, где тот расположен. В данном примере компонент находится в файле, на жестком диске моего компьютера Macintosh. Но что делать, если внешний компонент заархивирован где-нибудь в Internet или на другой машине? Возможно, компонент хранится в виде файла на рабочей станции с операционной системой Unix, где соглашения для имен файлов совсем другие, чем для операционной системы Mac.

При описании внешних компонентов можно определять для них *идентификатор общего доступа* (public identifier). Анализатор прежде всего попробует расшифровать идентификатор, и только в том случае, когда не сможет выполнить эту операцию, обратится к конкретной ссылке на файл.

```
<!ENTITY   public.statement
       PUBLIC      "-//Suneidesis//TEXT Mission Statement //EN "
       "http://localhost/~tfj/company.mission.statement.xml"
>
```

Если вам удалось заметить в этой схеме слабое место, примите наши поздравления! Действительно, как с помощью идентификатора `PUBLIC` пользователю (и анализатору) получить ответ, где находится этот самый необходимый компонент? Программисты SGML-анализаторов решают эту задачу с помощью файла-каталога. Чтобы получить подробную информацию о файле-каталоге, следует разобраться с конкретным, работающим у вас анализатором.

Использование файла-каталога помогает скрыть системные идентификаторы от пользователей и, что гораздо важнее, от их документов. Вы просто включаете в каталог записи для каждого компонента. Туда же можно «упрятать» идентификаторы чего угодно: определений типа документа, описаний компонентов и так далее. Это облегчает обмен документами между пользователями. Им не нужно редактировать полученный документ, достаточно будет изменить только их собственный каталог.

Таким образом, необходимо просмотреть документацию используемого XML-анализатора на предмет того, реализована ли в нем функция каталога, и если да, то каким образом.

```
   -- ISO latin 1 entity set for HTML --
PUBLIC     "ISO 8879-1986//ENTITIES Added Latin 1//EN//HTML" ISOlat1.sgm

DOCTYPE wrox    wrox.dtd
```

В этом примере последняя строка извлекает из файла-каталога на моем компьютере имена файлов, которые будут использоваться моим анализатором для поиска нужного DTD. Аналогичным образом указано расположение некоторых символьных компонентов.

Группы компонентов

Обычно в SGML- и XML-сообществах принято собирать описания компонентов вместе и сохранять их в файле (то есть во внешнем компоненте). Такой подход заметно облегчает работу, если только не вводить рекурсию (когда описание компонента ссылается на компонент, содержащий это описание). В самом деле, при работе с несколькими документами иметь такие *группы компонентов* (entity sets) – истинное благо.

Эту идею вполне можно распространить на описания элементов и атрибутов и также поместить их в виде компонента. Большинство общедоступных DTD определены именно таким образом. Потом на эти описания ссылаются, конечно, как на группы элементов и атрибутов, а не как на группы компонентов.

Одним из наиболее часто встречающихся вариантов применения подобных групп является включение стандартных символьных компонентов Международной организации по стандартам (ISO), которая определила ряд групп компонентов для различных категорий (относящихся к ее собственным стандартам на наборы символов). На некоторых сайтах, например, на сайте:

http://www.schema.net/entities/

в свободном доступе имеются XML-совместимые копии этих групп компонентов, так что читатели на их основе могут создать свой собственный внешний компонент:

```
<!-- Это наш собственный внешний компонент, который ссылается на внешний набор,
описывающий символьные компоненты. -->
<!ENTITY   ISOsets SYSTEM    "/home/XML/Entities/all.entities">
```

Теперь для ссылки на эту группу компонентов можно употреблять обозначение `&ISOsets;`. В таком случае анализатор будет просматривать идентификатор `PUBLIC` или считывать файл операционной системы, где обнаружит следующую запись:

```
<!-- Описываем компоненты, соответствующие каждому из наборов компонентов ISO. -->
<!ENTITY ISOlat1 PUBLIC"ISO 8879-1986//ENTITIES Added Latin 1//EN" "">
<!ENTITY ISOlat2 PUBLIC"ISO 8879-1986//ENTITIES Added Latin 2//EN" "">
<!ENTITY ISOnum PUBLIC "ISO 8879-1986//ENTITIES Numeric and Special
   Graphics//EN"   "">
...
<!-- Теперь ссылаемся на них, в результате чего (символьные) компоненты, которые,
в свою очередь, описаны в них, будут нам доступны. -->
&ISOlat1;
&ISOlat2;
&ISOnum;
...
```

Полный набор символов ISO определяется двадцать одной группой компонентов (табл. 2.6). Реальное содержание этих групп основано на списках символов, заданных в первоначальном стандарте SGML и связанных с этим стандартом технических отчетах.

Таблица 2.6. Наборы символов ISO

Название группы компонентов	Символьные компоненты
ISOlat1	Латинские буквы (то же, что и ASCII)
ISOlat2	Латинские буквы (дополнительные)
ISOgrk1	Современные греческие буквы
ISOgrk2	Современные греческие буквы
ISOgrk3	Греческие символы, применяемые в математике
ISOgrk4	Греческие символы, применяемые в математике
ISOcyr1	Современные русские буквы (кириллица)
ISOcyr2	Современные нерусские буквы (кириллица)
ISOdia	Диакритические знаки (ударения и т.д.)
ISOpub	Издательские символы
ISOtech	Технические символы
ISObox	Символы для рисования прямоугольников
ISOnum	Математические символы (численные)
ISOamsa	Математические символы
ISOamsb	Математические символы
ISOamsn	Математические символы
ISOamso	Математические символы
ISOamsr	Математические символы
ISOmfrk	Математические символы
ISOmopf	Математические символы
ISOmscr	Математические символы
ISOchem	Химические символы

Команды приложений

Следует честно признать, что существуют ситуации, когда язык XML оказывается недостаточно гибок и его возможностей может не хватить для создания особо сложных документов. Однако в XML существует механизм, позволяющий расширить функциональность языка и прибегнуть к помощи других приложений. Их можно активизировать, используя так называемые *команды приложений* (processing instructions, PI), которые, в отличие от тэгов и описаний разметки, ограничены сочетаниями символов <? и ?>.

Уточнение *Компании-разработчики программного обеспечения, использующие XML, часто употребляют команды приложений для включения информации или команд, характерных для разработанных ими приложений. Впоследствии такие XML-документы могут быть использованы в иных условиях, при этом специфическая информация разработчиков не будет влиять на использование документа и просмотр его содержания, поскольку в отсутствие соответствующего приложения в системе она бессмысленна.*

Команды приложения имеют вид:

```
<?PITarget processing instructions ?>
```

`PITarget` представляет собой имя приложения, которому и поручается произвести операции, предусмотренные отданной программе-приложению командой. Это имя может быть формально объявлено в описании `NOTATION`, причем оно не должно начинаться с последовательности символов `xml` в любой комбинации заглавных и строчных букв. Например, мы могли бы поместить описание `NOTATION` в определение типа документа:

```
<!NOTATION MyFabApp SYSTEM "file://mydir/FabApp.exe" >
```

Затем необходимо использовать следующую команду приложения в некоторой точке внутри документа, чтобы в процессе обработки документа передать команды в `FabApp.exe`:

```
<?MyFabApp  Do_this ?>
```

Программа `FabApp.exe` может быть приложением, использующим XML-процессор или анализатор, с целью извлечь данные из XML-файла. Она может быть также каким-либо другим приложением, выполняющим некоторую обработку, связанную с разбираемым документом.

Команда приложения позволяет передавать приложению и дополнительные данные. Стандарт XML предполагает, что это будут опции командной строки и другие команды, относящиеся к данному приложению.

Формат данных команд приложений в рамках стандарта XML не определен, за исключением команды `<?xml...?>` и тех команд, которые используют стиль «атрибут-значение». Команды (данные PI) приложению `PITarget` должны присутствовать в таком формате, который оно способно интерпретировать. Однако различные XML-анализаторы могут обращаться с PI-данными различными способами, поэтому следует проверить документацию вашего анализатора на предмет того, как он обрабатывает команды приложений.

Строго говоря, не будет ошибкой использовать описание `NOTATION` и применить соответствующую команду приложения, но если нужного приложения нет, то включение команды просто ни к чему. Дело разработчика – обеспечить наличие в системе необходимых приложений.

Есть одно небольшое, но существенное различие между тем, как обрабатываются команды приложений в языках XML и SGML. В XML данные PI заключаются в скобки вида `<?` и `?>`, а в SGML они по умолчанию заключаются в скобки вида `<?` и только `>`. Если документы должны обрабатываться как XML-, так и SGML-анализаторами, необходимо убедиться, что дополнительный знак вопроса не создаст проблем для приложений, основанных на SGML. Анализатор SGML воспримет знак вопроса как данные, которые должны быть переданы приложению, в то время как анализатор XML употребит его сам. Если изменить системное описание для анализатора SGML таким образом, чтобы закрывающий символ команды программе-приложению в SGML (Processing Instruction Close, PIC) стал `"?>"`, то эта проблема будет снята.

Описание текста в XML

Мы уже встречались с объявлением XML `<?xml version="1.0"?>`, которое обязательно появляется в начале документа. Оно также является *описанием текста XML* (XML text declaration). В данный момент оно зарезервировано на будущее, но одно из его применений уже определено.

Внешние компоненты, на которые мы ссылаемся в нашем документе, могут иметь различную кодировку символов; это справедливо и для внешнего подмножества нашего DTD. Внешние компоненты (отдельные XML-файлы или файлы с текстовыми данными) желательно всегда создавать в соответствии со стандартом кодировки Unicode UCS-2 (16 бит), но невозможно гарантировать, что все остальные пользователи последуют этому примеру. Описание текста в XML указывает способ, с помощью которого можно определить, какая кодировка использована в документе.

```
<?xml encoding="ISO-10646-UCS-2" ?>
```

Этот оператор должен располагаться в начале файла, содержащего описания внешних компонентов.

Если включить атрибут кодировки `encoding` в объявление XML для главного документа, как это сделано здесь:

```
<?xml version="1.0"? encoding="UTF-8" standalone="no">
<!DOCTYPE rootname SYSTEM "mydtd.dtd" [
<!ENTITY % xTypeParameterEntity SYSTEM "entdeclars.ent" >
%xTypeParameterEntity;
]>
```

то он влияет на весь документ, и каждый внешний компонент в другой кодировке должен иметь описание текста XML. Другими словами, если в приведенном примере файл с описаниями компонентов `entdeclars.ent` имел бы другую кодировку, описание текста XML в самом начале должно принять следующий вид:

```
<?xml encoding="EncodingName"?>
<!ENTITY ...>
<!ENTITY ...>
```

Существует много кодировок, имеющих различные названия, которые можно включить в описание XML-текста. Особое внимание стандарт XML обращает на те из них, которые соответствуют Unicode и его эквиваленту – стандарту ISO (ISO 10646). Эти кодировки управляют операцией распознавания 16-битовых или 8-битовых кодов символов, а также проверяют, должны ли в том или ином случае применяться какие-либо правила преобразования Unicode. Стандарт XML даже позволяет использовать другие схемы кодирования. Если подобная схема зарегистрирована в IANA (Internet Assigned Naming Authority), рекомендуется использовать приведенные там имена (об этом будет сказано ниже). Не исключен вариант использования даже тех документов, которые были подготовлены с помощью кодирования EBCDIC.

Совет *Дальнейшую информацию о IANA и названиях наборов символов можно узнать по адресу: http://www.iana.org/numbers.html. Там же можно найти дополнительные сведения о наборах символов (Character Sets).*

Когда компоненты создаются в нестандартных кодировках, название способа кодирования в описании текста XML должно совпадать с названием кодировки содержания. Само описание текста обязано соответствовать той же кодировке, которую оно объявляет.

Условные разделы

Меняются времена, меняются и задачи. Это касается и наших DTD, которые приходится постоянно редактировать, чтобы их можно было использовать в самых разнообразных целях. То же самое порой необходимо проделывать и с некоторыми описаниями, ведь в каких-то случаях они будут удовлетворять нашим требованиям, а в других нет. Спасением в таких ситуациях являются условные разделы, с помощью которых можно приводить в действие или дезактивировать выбранные части.

Уточнение *Условные разделы (conditional sections) могут быть использованы только во внешнем подмножестве DTD.*

Сначала мы рассмотрим два типа условных разделов: `INCLUDE` и `IGNORE`, а затем познакомимся с тем, как они работают.

Разделы INCLUDE

Раздел `INCLUDE` содержит описания в следующем виде:

```
<![INCLUDE[
<!-- Поскольку это общедоступный документ, включить все наборы компонентов ISO. -->
%ISOsets;
<!ELEMENT  included.element  (...)>
<!ATTLIST  included.element  ...>
]]>
```

Заметим, что все описания внутри секции `INCLUDE` имеют тот же синтаксис, что и находящиеся внутри DTD.

Разделы IGNORE

Разделы `IGNORE` анализируются, но никаких действий в отношении описания разметки в них не предпринимается.

```
<![IGNORE[
<!-- Не будем беспокоиться о наборах ISO, позже мы составим свои собственные. -->
%ISOsets;
<!ELEMENT  ignored.element   (...)>
<!ATTLIST  ignored.element   ...>
]]>
```

Теперь посмотрим, как их можно использовать.

Параметрические компоненты в описаниях условных разделов

Использование параметрических компонентов для включения или выключения секций `INCLUDE` и `IGNORE` является обычной практикой в кругу XML- и SGML-сообществ. Стандарт XML позволяет использовать ссылки на параметрический компонент там, где должны стоять ключевые слова `INCLUDE` и `IGNORE`.

```
<!ENTITY % confidential "IGNORE">
<!ENTITY % public  "INCLUDE">

<![%confidential;[
<!-- Данные описания элемента, атрибута и прочие используются только тогда, когда
в документе имеются конфиденциальные материалы.. -->
<!ELEMENT  sec.level  (#PCDATA, secret_stuff)>
]]>

<![%public;[
<!-- ..в то время как эти описания включаются только для общедоступных
документов. -->
<!ELEMENT  kick.back.rate    (#PCDATA) >
]]>
```

Когда приходится иметь дело с группой XML-документов, обработку которых необходимо провести в соответствии со схемой определений, расположенной внутри общедоступного раздела `%public` (но не в соответствии со схемой определений, приведенной в разделе `%confidential`), в описании компонента `%public` надо установить значение `"INCLUDE"`, а в описании компонента `%confidential` (конфиденциально) значение `"IGNORE"`. Если в будущем понадобится обработать документы в соответствии со схемой, приведенной в разделе `%confidential`, а не `%public`, достаточно поменять местами значения в описаниях компонентов. Если эти две части DTD хорошо спланированы, вам не придется редактировать и переписывать ключевые области во всем DTD.

Стиль описания

Прежде чем закончить разговор о структуре и приемах использования описаний, нам стоит взглянуть на них еще раз, но под другим углом зрения. Как их лучше всего располагать, чтобы любой, кому впоследствии придется читать наше определение типа документа, мог легко разобрать, какие структуры в него включены? Создавать описания, трудные для восприятия, это то же самое, что воспроизводить в документе замысловатую разметку. Есть ли смысл тратить время на то, чтобы усложнять работу других пользователей?

Во второй главе читатели познакомились со способом, которому автор этого раздела отдает предпочтение при размещении последовательности описаний XML. Я считаю такое расположение удобным, и надеюсь, что, когда мы разбирали примеры, оно и вам показалось вполне приемлемым.

Поработав некоторое время с XML, вы, возможно, захотите перейти на другой набор соглашений о расположении описаний. При этом не забывайте, что

анализатору XML безразлично, сколько пробелов помещается между словами в описаниях. Однако для вас, когда вы вернетесь к своему DTD, скажем, через шесть месяцев, это обстоятельство может оказаться весьма существенным. Вот основные правила, которые помогут вам в подобном случае:

1. Каждое описание должно быть расположено на отдельной строке:

```
<!ELEMENT e1 (...)>
<!ELEMENT e2 (...)>
```

   а не выстраиваться подряд в цепочку:

```
<!ELEMENTe1  (...)><!ELEMENT        e2       (...)>
```

2. Описание элемента должно предшествовать любому связанному с ним списку атрибутов:

```
<!ELEMENT e1 (...)>
<!ATTLIST e1 ...>
```

   а не следовать за списком атрибутов:

```
<!ATTLIST e1 ...>
<!ELEMENT e1 (...)>.
```

3. Не скупитесь на комментарии:

```
<!-- Комментариев не бывает слишком много. -->
```

4. Комментарии, описывающие предназначение DTD, надо размещать в самом начале;

5. Не обрамляйте длинные комментарии многочисленными символами; используйте только символы **<!--** и **-->** и пробельные литеры в достаточном количестве:

```
<!--

Предпочтительный формат для длинных комментариев

-->
```

   вместо того чтобы делать так:

```
<!---------------------------
- -
- Плохой формат для длинных комментариев -
- -
--------------------------->
```

6. И самое главное правило: *будьте последовательны*!

Заключение

В этой главе мы рассмотрели основную спецификацию XML 1.0, а также, в соответствии с определением данной спецификации, различия между правильными и состоятельными документами. Большая часть второй главы была посвяще-

на синтаксису XML, однако мы надеемся, что, кроме того, вы получили некоторое представление о достоинствах XML. К настоящему моменту вам следует:

- знать, как размечать документы, используя XML;
- иметь понятие о гибкости XML, вытекающей из возможности создания собственных тэгов;
- разбираться в том, как описываются в DTD элементы, атрибуты, компоненты и другие части состоятельных документов.

В последующих главах мы более подробно рассмотрим материал, изученный в этой главе, чтобы дать вам серьезные знания и опыт в этой быстро развивающейся и впечатляющей области технологий Internet.

Глава 3. XML-схемы

В соответствии с общепринятым определением, схема – это обобщенный план или диаграмма. В информатике схемы определяют характеристики классов объектов. Применительно к XML это означает, что схемы описывают способ разметки данных, и поэтому классическое определение типа документа (DTD) тоже является схемой.

В начале этой главы будет уместно дать общие сведения о схемах, далее мы затронем вопрос об ограничениях и определенных недоработках, связанных с применением определения типа документа в XML. Затем читатели познакомятся с двумя новыми проектами, которые используют синтаксис XML для описания структуры документа и в настоящий период находятся в разработке.

Третья глава как раз посвящена изучению этих двух методик. Называются они *XML-данные* (XML-Data) и *описание содержания документа для XML* (Document Content Description for XML, DCD). Во время подготовки книги к изданию обе из них еще находились на стадии сообщений в консорциуме W3C. Группой XML-Dev был предложен еще один способ разметки данных, получивший название *схема для объектно-ориентированного XML* (Schema for Object-oriented XML, SOX), но ее мы рассматривать не будем, несмотря на то, что эта методика уже частично внедрена в некоторое свободно распространяемое программное обеспечение. Ссылки на все схемы даны в «Приложении B».

В октябре 1998 года, к моменту, когда наш сборник уже находился в печати, в консорциуме W3C была наконец сформирована рабочая группа по XML-схемам. Ее участникам было поручено разработать наиболее приемлемую для всех пользователей методику, написанную в синтаксисе XML. Ожидается, что в первую очередь группа проанализирует современное состояние исследований в области XML-данных и DCD, затем приступит к изучению представленных предложений, касающихся способов разметки, основанных на языке XML. В результате должен появиться особый документ, где участники группы, по-видимому, сформулируют основные темы, которые будут предложены XML-сообществу для дальнейшего обсуждения, а также рамки самой дискуссии. Другими словами, не исключена вероятность, что какое-либо из этих двух предложений, которым будет посвящена глава 3, в конце концов будет отвергнуто. Однако обзор их содержания дает прекрасную возможность ввести читателей в область информативного проектирования,

детально изучить принципы построения схем, основанных на XML, ведь какой-то вариант в самое ближайшее время будет наверняка принят. Это должна быть лучшая из всех возможных версия, на основе которой после ее официального утверждения станут создаваться новые приложения, использующие XML. (Отметим, что в данном случае приемлем термин «версии», так как не исключена вероятность, что подобных методик будет несколько.)

Прежде чем приступать к изучению той или иной схемы, следует хотя бы в общих чертах поговорить об их назначении. Схемы тесно взаимосвязаны с теми целями, которые имелись в виду при создании самого языка XML, а именно:

- этот язык должен быть пригоден для непосредственного использования в Internet;
- XML должен поддерживать разнообразные приложения;
- XML должен быть совместим с SGML;
- написание программ для обработки XML-документов не должно вызывать особых затруднений.

Короче говоря, задача состоит в том, чтобы сделать XML языком, пригодным (и удобным) для обмена информацией между приложениями. С точки зрения профессионалов, работающих в прикладных областях и использующих специальные программы, любая подобная схема обязана накладывать определенные ограничения на XML-документ. Это позволит создавать программное обеспечение, которое, с одной стороны, заранее «знает», какую информацию оно может ожидать, а с другой – вводить в проекте программы допущения, которые помогут расширить возможности приложения. При этом проверка на состоятельность окажется в то же время и проверкой на соответствие документа его схеме, что позволит быстро удостовериться, способно ли приложение принять документ к исполнению и выполнить запрос, используя при этом все свои возможности.

С точки зрения пользователя, проверка состоятельности по отношению к схеме увеличивает шансы на то, что передаваемая информация будет понята и употреблена с пользой. Отправитель может оценить правильность своего сообщения до того, как оно будет послано, а адресат прежде, чем его обрабатывать, испытает его на состоятельность.

Усиливающийся интерес к XML-схемам обусловлен в первую очередь потребностями в обмене новыми типами приложений, для создания которых используется описываемый здесь язык. Несмотря на то, что за долгие годы эксплуатации в этом проекте были обнаружены некоторые пробелы и ограничения, определения типа документа SGML/XML до сих пор во многом удовлетворяет пользователей. Как было сказано выше, обмен документами в основном связан с их структурой и иерархией, которые достаточно подробно описываются в определении типа документа. Однако возрастающий поток обмена данными между обрабатывающими приложениями требует более строгой проверки их на состоятельность. Возможность как можно раньше накладывать ограничения на применяемые в подобных материалах типы данных, а также обеспечение легкой расширяемости и обработки документов дорогого стоят. Как раз об этих вопросах следует поговорить подробнее.

Определение типа документа XML как схема

Как уже было сказано при разборе XML-документов, схема является описанием способа разметки документа: его грамматики, словаря, структуры, типов данных и т.д. Классическое определение типа документа тоже является схемой для XML, оно имеет свои преимущества и недостатки.

Читатели уже знают, как создается определение типа документа XML, при этом они, возможно, уже догадались, что для описания разметки в настоящее время, к сожалению, используется синтаксис, отличный от синтаксиса XML. Этот синтаксис обычно называют расширенной формой Бэкуса-Наура, или EBNF.

Схемы типа DTD имеют и несколько других, менее очевидных недостатков. Вот лишь несколько из них (к другим мы вернемся позже и подробно рассмотрим каждый пункт):

- создать удобные для работы DTD достаточно трудно;
- DTD не расширяемы;
- DTD плохо описывают XML как данные;
- не обеспечивается наследование свойств из одного DTD в другое;
- DTD не поддерживает пространства имен;
- описательная способность DTD ограничена;
- нет механизма, обеспечивающего содержание элемента по умолчанию;
- авторская работа по созданию программного продукта фактически разбивается на два этапа: первый заключается в создании DTD, второй – в формировании XML-документа. Таким образом, для обработки документа нужно иметь два анализатора.

Вот почему особый интерес представляет использование альтернативных схем, особенно тех, которые основаны на синтаксисе XML.

Одно из таких предложений, представленных в консорциум W3C в январе 1998 года, содержится в сообщении по XML-данным, которое можно найти по адресу:

http://www.w3c.org/TR/1998/NOTE-XML-data

Другое предложение, представленное в июле 1998 года, находится в сообщении об описании содержания документа для XML. Его адрес:

http://www.w3c.org/TR/NOTE-dcd

Синтаксис в сообщении о DCD по форме похож на синтаксис, описанный в сообщении об XML-данным, но включает также часть синтаксиса RDF.

Есть еще одно предложение, слишком поздно представленное и по этой причине не вошедшее в нашу книгу. Оно поступило в октябре 1998 года. Его назвали *схемой для объектно-ориентированного XML* (SOX), найти ее можно по адресу:

http://www.w3c.org/Submissions/1998/15/

К сожалению, нет никаких гарантий, что окончательная конфигурация схем, использующих синтаксис XML, будет сродни какому-либо из этих двух главных сообщений. Однако окончательная, приемлемая для большинства пользователей

схема или схемы, основанные на XML, неизбежно скорее рано, чем поздно, появятся на свет, и в любом случае принципы их построения будут точно такими же, как и в упомянутых выше проектах. Так что знакомство с ними позволит вам легко перейти к любому новому синтаксису.

В этой главе в первую очередь предлагается дать общий анализ схем, основанных на XML (XML-данные и DCD для XML). Затем будут рассмотрены некоторые вопросы, касающиеся схем вообще, затем уже мы вплотную займемся XML-данными и, наконец, DCD.

XML-схема: общие вопросы

Трудности написания хорошего DTD

Определение типа документа создавать трудно. Точнее, как уже было сказано в главе 2, непросто написать *хорошее* определение типа документа, поэтому, приступая к новому проекту, в котором планируется использовать языки разметки, программисты стараются найти готовое DTD, применимое к большей его части. Отличная идея – взять из ранее написанного DTD все необходимое, а ненужные фрагменты выбросить! Пространства имен, например, позволяют ссылаться на конкретные элементы в различных определениях типов документов. Беда, однако, в том, что в классической теории DTD до сих пор не существует способа, с помощью которого можно было бы взять раздел `<Author>` (автор) из определения типа документа с названием `Book` (книга) и раздел `<content>` (содержание) из некоторого гипотетического определения, относящегося к издательству `Wrox`, затем совместить их, проверить результат на правильность и получить в точности то, что требуется.

Тот факт, что DTD не использует синтаксис XML, еще более затрудняет его описание. Как было сказано ранее, в DTD применяется форма обозначений, называемая EBNF, которая когда-то на меня лично просто наводила страх. Со временем, много раз вынужденно обращаясь к этой методике, я приобрел вкус к синтаксису EBNF, теперь он мне уже почти нравится, иногда я даже начинаю превозносить его достоинства. Но тем из вас, кто предпочитает синтаксис попроще, лучше использовать XML-данные или DCD.

Нерасширяемость DTD

Если бы мне потребовалось оформить книжный каталог и под него создать DTD под названием `book`, а потом вдруг возникла необходимость добавить новый раздел в код для книг по ASP, мне пришлось бы, как вы, вероятно, уже догадались, переписывать все определение типа документа. И это только потому, что DTD не расширяемы. Но этим бы дело не ограничилось – я был бы вынужден также удостовериться в том, что первоначальный и новый документы остались состоятельными.

DTD плохо описывает данные XML

Первоначально XML и DTD, так же как и SGML, виделись как структурированный синтаксис документов. Теперь XML все больше и больше рассматривают

как способ хранения данных и объектов. Чтобы понять, в чем тут дело, обратимся к примеру.

Разберем XML-файл, который размечает класс объектов, называемый `hat` (шляпа).

```
<class name= "hat">

   <hat id="h1">
       <style>Baseball</style>
       <size>8</size>
       <color>red</color>
   </hat>

   <hat id="h2">
       <style>Stetson</style>
       <size>7</size>
       <color>gray</color>
   </hat>

   <hat id="h3">
       <style>Bowler></style>
       <size>6</size>
       <color>Black</color>
   </hat>

</class>
```

Теперь самое время вспомнить, что было написанно во введении к первой главе о классах и объектах. В приведенном выше описании имеется три объекта (на которые можно сослаться с помощью их номера `id` как на 1, 2 и 3) класса `hat` с различными свойствами (`style`, `size` и `color`). Кроме того, в описании есть и значения этих свойств.

В терминологии Java имя класса – `hat`, и каждый элемент `<hat>` является частным случаем класса `hat` и объектом `hat`. Дочерние элементы `<style>`, `<size>` и `<color>` являются атрибутами класса вместе с их значениями, заключенными между тэгами.

Если же взглянуть по-иному, то каждый объект `hat` является частным случаем класса `hat`, и каждый объект `hat` имеет свойства, определенные атрибутами класса.

Документ, приведенный выше, мог бы иметь следующую схему (классическое DTD):

```
<!DOCTYPE class [

<!-- Описания элементов. -->

<!ELEMENT class (hat)>
<!ELEMENT hat (style,size,color)>
<!ELEMENT style (#PCDATA)>
<!ELEMENT size (#PCDATA)>
<!ELEMENT color (#PCDATA)>
```

```
<!-- Описания атрибутов. -->
<!ATTLIST hat id ID #REQUIRED>

<!-- Предопределенные компоненты. -->

<!ENTITY lt"&#38;#60;">
<!ENTITY gt"#62;">
<!ENTITY amp    "&#38;#38;">
<!ENTITY apos   "&#39;">
<!ENTITY qout   "&#34;">
<!-- Конец DTD. -->
```

Это DTD прекрасно удовлетворяет требованиям схемы документа, однако в качестве схемы разметки данных оно имеет ряд серьезных ограничений. Вот некоторые из наиболее очевидных:

- для конкретного DTD не существует способа расширения класса. Для этого придется писать новое DTD;
- не существует способа, с помощью которого класс `hat` (шляпа) мог бы наследовать свойства других классов, например, класса `clothes` (одежда);
- не существует способа, с помощью которого некоторый дочерний класс, например класс `female-hat` (дамская шляпа) мог бы наследовать свойства класса `hat`;
- не существует подразделения данных на типы. Размер `(size)` шляпы не является целым числом: в данном документе он является строкой, и останется ею до тех пор, пока мы будем использовать DTD для описания класса `hat`.

DTD не поддерживает пространства имен

Чтобы ввести тип элемента в документ, можно использовать пространство имен, однако этого нельзя делать, когда в DTD требуется сослаться на описание элемента или объекта. И, если пространства имен все же используются, возникает необходимость при включении в него любых элементов, взятых из других пространств имен, модифицировать DTD. (Пространства имен будут подробно рассмотрены в следующей главе.)

Ограничения описательной способности DTD

DTD имеет очень строгий формат, который вполне пригоден для определения структур документа. Тем не менее, для других приложений может потребоваться больше информации об элементе. Для этого в схеме DCD служит элемент `description`.

DTD и содержание элемента по умолчанию

Несмотря на то, что ни одна из схем, которые будут рассмотрены в данной главе, не обеспечивает содержание элемента по умолчанию (это делает язык XSL, описанный в главе 8), можно представить себе схему, которая способна добавлять в элемент содержание по умолчанию, например, какой-нибудь шаблонный текст или вежливый отказ.

Проверка определения типа документа

Если у вас есть браузер IE5, можно предложить вам простой HTML-файл, который позволяет анализировать и проверять на правильность XML-файлы и DTD. Вы можете найти его по адресу http://webdev.wrox.com/books/1525/, он называется `xmltest.htm`. Вот код этого файла:

```
<HTML>
<HEAD>
<TITLE>Checking XML Files for Errors</TITLE>
<STYLE TYPE="text/css">
BODY {font-family:Tahoma,Arial,sans-serif; font-size:10pt;}
</STYLE>

<SCRIPT>

function startTest(strXMLFile) {

// Создадим объект activeX.
var myDocument = new ActiveXObject("microsoft.xmldom");

// Загрузим xml-файл в объект activeX и назовем его myDocument.
myDocument.async=false;
myDocument.load(strXMLFile);

// Построим строку подробного сообщения об ошибках.
strErr = new String();

strErr = 'Source URL: <B>'
       + myDocument.parseError.url
       + '</B><P>';

strErr += 'Error Description: <B>'
       + myDocument.parseError.reason
       + '</B><BR>';

strErr += 'Error Number: <B>'
       + myDocument.parseError.errorCode
       + '</B><BR>';

strErr += 'Error File Position: <B>'
       + myDocument.parseError.filepos
       + '</B><BR>';

strErr += 'Error Line Number: <B>'
       + myDocument.parseError.line
       + '</B><BR>';

strErr += 'Error Line Character: <B>'
       + myDocument.parseError.linepos
       + '</B><BR>';

// Построим строки указателя ошибок.
strShow = new String();
strShow = myDocument.parseError.srcText;
```

```
strPoint = new String();
for (i = 1; i < myDocument.parseError.linepos; i++)
strPoint += '-';
strPoint += '^';

// Покажем строки на странице.
document.all['errText'].innerHTML = strErr;
document.all['errLine'].innerText = strShow;
document.all['errPoint'].innerText = strPoint;
myDocument = null;
}
</SCRIPT>

</HEAD>
<BODY>

<H4>Checking XML Files for Errors</H4>
<SPAN ID="errText"></SPAN><P>

<PRE>
<SPAN ID="errLine"></SPAN>
<SPAN ID="errPoint"></SPAN>
</PRE>

<DIV STYLE="position:absolute; top=250">
<HR>
<INPUT TYPE="BUTTON" VALUE="   " ONCLICK="startTest('wroxdb1.
xml')">
  Push the button to test your file.
</DIV>

</BODY>

</HTML>
```

На пятой строке от конца, где у нас написано:

```
ONCLICK="startTest('wroxdb1.xml')">
```

подставьте местонахождение (URL) документа XML, который вы хотите проверить. Файл HTML, чтобы найти ошибки в вашем документе, использует анализатор Microsoft XML. Ошибки будут отражены в окне предупреждений, и каждая из них должна быть исправлена отдельно.

Создание базы данных при помощи XML

Несмотря на то, что язык XML как средство создания базы данных будет подробно описан в главе 10, сейчас тоже необходимо уделить внимание этому вопросу, иначе читателям будет трудно разобраться, почему при употреблении XML в качестве системы хранения в реляционной базе данных нужны схемы, отличные от классического DTD.

В общем смысле база данных представляет собой любую совокупность сведений. Однако в обычном – «компьютерном» – применении база данных интерпретируется как совокупность родственной информации. В табл. 3.1приведены некоторые термины, которые используют при обсуждении баз данных.

Таблица 3.1. Терминология баз данных

Термин	Описание
Tables (таблицы)	Совокупность информации, которая по какой-либо логической причине была собрана вместе, например перечень частей чего-либо или список имен и адресов клиентов, как в примере, приведенном ниже
Records (записи)	Запись представляет собой отдельный элемент таблицы. Называется также строкой
Fields (поля)	Отдельные пункты, которые вместе составляют полную запись. Называются также столбцами
Index (указатель)	Поле или совокупность полей, которые используются для представления таблицы в некотором логическом порядке. В примере, приведенном ниже, поле customer name (имя клиента) является указателем
Primary key (первичный ключ)	Поле или совокупность полей в таблице, которые единственным образом определяют запись. Часто это число. В примере оно дано как customer id (идентификатор клиента). Иногда первичный ключ совпадает с указателем
Database engine (машина базы данных)	Программа, которая хранит в определенном порядке, организует и отыскивает всю информацию, содержащуюся в базе данных
Query (запрос)	Процесс получения желаемой информации из базы данных
Recordset (группа записей)	Совокупность различных записей или таблиц

Сейчас речь пойдет о том, каким способом можно представить в XML-документе простую таблицу (табл. 3.2).

Таблица 3.2. Простой пример XML-базы данных

Customer Name (Имя клиента)	Customer id (Идентификатор клиента)	Customer Address (Адрес клиента)	Customer Phone number (Телефон клиента)
Блогз, Джоан	1	24 Hemlock Place	216-222-3333
Смит, Джон	4	5 Maple street	440-123-4567
Смит, Джон	3	144 Elm	406-765-4321
Вильямс, Хью	2	27 Chestnut	257-757-5757

В таблице имеется четыре записи или строки, и четыре поля или столбца.

Полем указателя служит имя клиента (что вполне естественно), чтобы можно было расположить имена в алфавитном порядке, а полем первичного ключа является уникальное целое число, в данном случае идентификатор клиента (что тоже разумно, поскольку нам может встретиться несколько одинаковых имен).

Один из способов, с помощью которого эта база данных может быть записана в XML, выглядит следующим образом. Уверен, вы можете придумать и другие.

```
<customer-table>
   <customer-id id= "a001">
       <customer-name>Bloggs,Joan</customer-name>
       <customer-address>24 Hemlock Place </customer-address>
       <customer-phone>216-222-3333</customer-phone>
   </customer-id>
   <customer-id id= "a004">
       <customer-name>Smith, John </customer-name>
       <customer-address>5 Maple street </customer-address>
       <customer-phone>440-123-4567</customer-phone>
   </customer-id>
   <customer-id id= "a003">
       <customer-name>Smith, John </customer-name>
       <customer-address>144 Elm</customer-address>
       <customer-phone>406-765-4321</customer-phone>
   </customer-id>
   <customer-id id= "a002">
       <customer-name>Williams, Hugh </customer-name>
       <customer-address>27 Chestnut </customer-address>
       <customer-phone>257-757-5757</customer-phone>
   </customer-id>
</customer-table>
```

Совет *Обратите внимание, что идентификатор клиента в этом случае начинается с буквы. Дело в том, что значения атрибута id в языке XML не могут начинаться с цифры, а только с буквы или символа подчеркивания.*

В данном XML-файле мы сделали полем индекса идентификатор пользователя `customer_id` (вместо имени клиента, как это было в таблице). Уверен, теперь вы сможете без труда написать DTD для этого документа.

Однако если мы имеем набор таблиц, например таблиц заказов, возникают некоторые трудности.

```
<customer-order>
   <customer-id id= "a002">
   etc......
</customer-order>
```

Очевидно, и здесь, как и в таблице, приведенной выше, будут использованы те же данные, в частности, тот же самый уникальный идентификатор. Но как в определении типа документа отразить, что идентификатор в *таблице заказов* клиентов тот же самый, что и идентификатор в *таблице клиентов*?

Ответ на удивление прост – никак. Не существует способа соотнести друг с другом два документа и дать программному обеспечению XML информацию о том, что оба они являются частью одной группы записей. Вот почему так важно изучить

схемы XML-данных и DCD, которые способны увязывать подобные варианты.

При работе с небольшими базами данных недостатки DTD как схемы могут показаться тривиальными, однако когда приходится иметь дело с базами, содержащими миллионы записей и тысячи различных типов запросов, проблемы, возникающие из-за того, что определение типа документа под названием *x* нельзя соотнести с определением типа документа, обозначенным как *y*, становятся попросту губительными для дела.

XML-данные: предлагаемое решение

Заметим, что мы использовали термин «предлагаемое решение», поскольку в настоящее время эта схема еще даже не является рабочим проектом консорциума W3C.

Представленная схема способна справиться с описанными выше трудностями, кроме того, ее можно использовать в качестве определения типа документа. Когда XML-данные (или любая другая подобная схема) присоединяет к себе DTD, она становится *синтаксической схемой* (syntactic schema), то есть схемой, которая описывает синтаксис документа. Если к ней присоединить объект данных, она становится *концептуальной схемой* (conceptual schema), то есть схемой, которая описывает соотношения между концепциями или объектами.

Не беспокойтесь – вам не придется изучать два синтаксиса. В обоих случаях они одинаковы, различными должны быть подходы, а также ваше личное понимание проблемы.

Простейший пример XML-данных

Давайте начнем с несложного примера. Поскольку вы уже знакомы с концепцией DTD, продемонстрируем, как переделать простейшее определение типа документа в XML-данные. Опыт и традиции подсказывают начать с классической фразы «Здравствуй, мир!»

Перед вами правильный и состоятельный документ `"Здравствуй, мир!"`

```
<?xml version= "1.0" ?>
<!-- DTD -->
<!DOCTYPE greeting [
<!ELEMENT greeting (#PCDATA)>
]>

<greeting> Hello World! </greeting>
```

Теперь преобразуем этот документ в XML-данные.

Основной принцип работы схемы XML-данных заключается в создании описаний `elementType` (тип элемента), которые формулируют его в терминологии XML. (Или объект класса в концептуальной схеме). У описания `elementType` есть атрибут двойного назначения `id`: он не только идентифицирует определение, но и именует определяемый класс.

Итак, следующая запись:

```
<elementType id= "greeting"/>
```

идентифицирует тип <greeting> элемента и имеет то же назначение, что и запись:

```
<!ELEMENT greeting... >
```

в синтаксической схеме DTD.

Совет *Последняя запись идентифицирует также класс greeting, если мы используем наше DTD с документами, которые хранят данные, то есть если мы используем нашу схему в качестве концептуальной. Синтаксис тот же самый, и документ выглядит точно так же, несмотря на то, что назначение документа иное.*

Вторая часть описания нашего элемента в DTD выглядит следующим образом:

```
<!ELEMENT greeting (#PCDATA)>
```

Она сообщает анализатору, что элемент greeting может в качестве содержания иметь обрабатываемые символьные данные (например, неразмеченный текст), но только не элементы.

Как можно выразить этот замысел в схеме XML-данных? Очень просто – добавляя элемент <string/>. Итак, имеем:

```
<elementType id= "greeting">
<string/>
</elementType>
```

Эта запись выражает в точности то же самое, что и наше описание элемента в DTD.

Окончательный вид документа вместе с прологом будет следующим:

```
<?xml version= "1.0" ?>
<schema xmlns="urn:W3C.org:xmlschema"
   xmlns:dt="urn:W3C.org:xmldatatypes">
<elementType id= "greeting">
<string/>
</elementType>
</schema>

<greeting> Hello World! </greeting>
```

С первой строкой, объявлением XML, мы уже знакомы, однако возникает вопрос, что означают последующие строки. В схеме XML-данных все описания элементов должны находиться внутри элемента schema. Чтобы определить, какой тип схемы применен в этом случае, необходимо включить пространство имен в открывающий тэг <schema>. Для этого служит вторая строка примера:

```
<schema xmlns="urn:W3C.org:xmlschema"
```

Она описывает пространство имен XML, определяемое URN, которое должно быть включено, чтобы указать агенту, проверяющему на состоятельность, что для определения документа мы используем схему, называемую xmlschema. Пространс-

тва имен рассматриваются в главе 4. Что же касается третьей строки примера:

```
xmlns:dt="urn:W3C.or:xmldatatypes">
```

то она записывается для схемы, которая определяет типы данных, и очевидно, в таком простом документе может быть опущена, поскольку не возникает вопроса о состоятельности типа данных элемента <greeting>. Помещена она здесь исключительно для полноты. Эта строка не создаст дополнительной работы анализатору, поскольку тот не должен обращаться к URN до тех пор, пока предписанный префикс не будет связан с каким-либо элементом.

Закрывающий тэг </schema> должен быть помещен в конце описаний атрибутов и элементов. Для соответствия нашего документа схеме XML-данных необходимо использовать этот элемент именно в той форме, как здесь показано.

Уточнение *Приведенные примеры используют синтаксис пространств имен рабочего проекта от 2-ого августа 1998 года (смотрите изменения по адресу http://www.w3c.org/TR/), а не синтаксис спецификации XML-данных.*

Синтаксис в приведенном выше примере использует понятие *области действия* (scoping) (о нем будет сказано позже, в главе, посвященной пространствам имен); с его помощью можно включить в это пространство все дочерние элементы. Для некоторых пользователей, возможно, будет более удобен следующий состоятельный синтаксис:

```
<?xml version= "1.0" ?>
<schema xmlns:s="urn:W3C.org:xmlschema"
   xmlns:dt="urn:W3C.org:xmldatatypes">
<s:elementType id= "greeting">
<s:string/>
</s:elementType>
</schema>

<greeting> Hello World! </greeting>
```

Имейте в виду, если вы обработаете любой из приведенных выше примеров XML-анализатором, который не поддерживает схему XML-данных, он выдаст ошибку, поскольку для него подобный документ не является правильным. В нем нет уникального открывающего и закрывающего тэга. Чтобы сделать его правильным, достаточно добавить две указанные ниже строки в качестве открывающего и закрывающего тэгов. Эти строки можно удалить по окончании обработки примера анализатором, не поддерживающим XML-данные. Браузер, способный работать с XML-данными, конечно же, сразу распознает, что пространство имен представляет собой схему.

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

<root>

< s:schema xmlns:s="urn:W3C.org:xmlschema"
   xmlns:dt="urn:W3C.org:xmldatatypes">
```

```
   <s:elementType id= "greeting">
<s:string/>
   </s:elementType>
   </s:schema>

<greeting> Hello World! </greeting>

</root>
```

Теперь, когда мы познакомились с методикой преобразования DTD в XML-данные, рассмотрим более сложный пример.

Более сложный пример XML-данных

Помните, мы собирались построить в XML базу данных, перечисляющую книги, публикуемые издательством Wrox Press. Теперь давайте рассмотрим определение типа документа для этой базы данных и его вид в схеме XML-данных.

Шаблон XML файла `ch03_books.xml` выглядит следующим образом:

```
<wrox-books-table>

   <book id="">
       <title></title>
       <author rank=""></author>
       <isbn></isbn>
       <publication-date></publication-date>
       <book-type></book-type>
       <book-family></book-family>
   </book>

</wrox-books-table>
```

Далее приведено определение типа документа для этого шаблона:

```
<?xml version="1.0"?>
<!DOCTYPE wrox-books-table [

<!ELEMENT wrox-books-table (book)+>
<!ELEMENT book (title,author+,isbn,publication-date,book-type,book-family+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT publication-date (#PCDATA)>
<!ELEMENT book-type (#PCDATA)>
<!ELEMENT book-family (#PCDATA)>
<!-- Список атрибутов. -->

<!ATTLIST book
   id ID #REQUIRED>
<!ATTLIST author
   rank CDATA #IMPLIED>
]>
```

Если обработать этот пример с помощью приведенного ранее теста, который хранится в файле `xmltest.htm`, программа попросит ввести значение для атрибута

id, поскольку в DTD объявлено, что это значение обязательно.

Версия файла ch03_books.xml, находящаяся на нашем сайте, как раз содержит это DTD. Если же вы вводите пример вручную, вам придется перед тестированием включить DTD в данный файл. Напоминаем, что вы можете найти файл xmltest.htm, так же как и все другие примеры этой главы (и книги), на нашем Web-сайте http://webdev.wrox.co.uk/books/1525/.

Рассмотрим, как можно записать приведенное DTD в синтаксисе XML-данных.

Описание элементов в XML-данных

При выполнении этой операции в первую очередь следует описать типы имеющихся элементов. Это можно сделать с помощью записи elementType и атрибута id, указывающего имя элемента нашего XML-документа.

```
<?xml version= "1.0" ?>
< schema xmlns="urn:W3C.org:xmlschema"
   xmlns:dt="urn:W3C.org:xmldatatypes">
<elementType id= "wrox-books-table"/>
<elementType id= "book"/>
<elementType id= "title"/>
<elementType id= "author"/>
<elementType id= "isbn"/>
<elementType id= "publication-date"/>
<elementType id= "book-type"/>
<elementType id= "book-family"/>

</schema>
```

В нашем определении типа документа описание элемента wrox-books-table указывает, что этот элемент может иметь один или более элементов book, а именно:

```
<!ELEMENT wrox-books-table (book)+>
```

Возникает вопрос, как, используя XML-данные, выполнить это требование?

Описание содержания элемента в XML-данных

Вернемся к описанию PCDATA в нашем примере "Здравствуй, мир!", а именно:

```
<elementType id= "greeting">
<string/>
</elementType>
```

Для описания включения элемента мы используем элемент element как дочерний по отношению к elementType:

```
<elementType id= "wrox-books-table">
   <element type="#book" occurs="ONEORMORE"/>
</elementType>
```

Обратите внимание на синтаксис данного примера. Элемент element имеет два атрибута: атрибут type (тип) и атрибут occurs (сколько раз встречается). Атрибут type принимает значение имени элемента (в данном случае элемента <book>), которому предшествует знак #.

Элемент element является дочерним к элементу <elementType id="wrox-books-table">, и потому его тип (type) – в данном случае элемент <book> – должен быть включен как дочерний в XML-документ.

Конечно же, элемент book имеет свои собственные дочерние элементы, и включение элемента book в wrox-books-table также подразумевает включение всех потомков book.

Атрибут occurs принимает значение строки, в рассматриваемом случае строки "ONEORMORE", которая эквивалентна знаку плюс (+) в классическом обозначении EBNF. Как интерпретируются другие обозначения, встречающиеся в примере, показано в табл. 3.3.

Таблица 3.3. Перевод нотации EBNF в нотацию XML-данных

Обозначение EBNF	Обозначение XML-данных	Примечания
Нет	REQUIRED (обязательный)	*)
? (знак вопроса)	OPTIONAL (необязательный)	
* (звездочка)	ZEROORMORE (ноль или более)	
+ (плюс)	ONEORMORE (один или более)	

*) Как и в синтаксисе классического DTD, значение REQUIRED используется по умолчанию. То есть если ничего не оговаривается, должен быть один и только один элемент.

Ниже приведено описание элемента book. Обратите внимание: что, несмотря на то, что для элемента title в качестве примера определено occurs="REQUIRED", такое же значение для других элементов не определено, поскольку оно присваивается по умолчанию.

```
<elementType id= "book">
   <element type="#title" occurs="REQUIRED"/>
   <element type="#author" occurs="ONEORMORE"/>
   <element type="#isbn"/>
   <element type="#publication-date"/>
   <element type="#book-type"/>
   <element type="#book-family" occurs="ONEORMORE"/>
</elementType>
```

Определение порядка следования элементов

Вследствие того, что в определении типа документа в качестве оператора используется запятая, элементы в нем должны выстраиваться в той же очередности, в какой они определены. В XML-данных ту же роль играет сам порядок, в котором элементы появляются как потомки элемента elementType.

В классическом DTD можно нарушить порядок следования элементов, используя в качестве маркера символ «вертикальная черта», за которым следует индикатор «звездочка» .

Например, мы могли бы написать:

```
<!ELEMENT book (title | author | isbn | publication-date | book-type | book-
family)*>
```

что позволило бы включать любое число указанных элементов в любом порядке.

В XML-данных тоже разрешено нарушить порядок следования элементов и с помощью этого приема удается получить дополнительные возможности, не предусмотренные в DTD. При этом используется элемент `group` (группа).

```
<elementType id= "book">
   <group occurs= "OPTIONAL">
       <element type="#title" occurs="REQUIRED"/>
       <element type="#author" occurs="ONEORMORE"/>
       <element type="#isbn"/>
       <element type="#publication-date"/>
       <element type="#book-type"/>
       <element type="#book-family" occurs="ONEORMORE"/>
   </group>
</elementType>
```

Эта запись означает: в случае появления одного из таких элементов следует ожидать, что обнаружится весь ряд. Другими словами, либо нет ни одного элемента, либо они должны появиться все сразу, однако порядок следования элементов не произволен.

Мы можем нарушить порядок, используя атрибут `groupOrder` (порядок в группе) элемента `group`. Атрибут `groupOrder` может принимать значения `"SEQ"`, `"AND"` или `"OR"` (табл. 3.4).

Таблица 3.4. Управление порядком следования элементов в схеме XML-данных

SEQ	Значение по умолчанию, которое просто обеспечивает, что элементы будут появляться именно в том порядке, в каком они перечислены. Обычно его опускают
AND	Нарушает порядок элементов и позволяет им появляться в любом порядке
OR	Позволяет появляться любому элементу из списка элементов (или групп)

Если бы атрибут `groupOrder` был включен, то пример, приведенный выше, начинался бы так:

```
<elementType id= "book">
   <group groupOrder="SEQ" occurs="OPTIONAL">
...etc.
```

Произвольный порядок определяется в следующем примере:

```
<elementType id= "book">
   <group groupOrder="AND" occurs="OPTIONAL">
       <element type="#title" occurs="REQUIRED"/>
       <element type="#author" occurs="ONEORMORE"/>
       <element type="#isbn"/>
       <element type="#publication-date"/>
       <element type="#book-type"/>
       <element type="#book-family" occurs="ONEORMORE"/>
   </group>
</elementType>
```

Это означает, что в процессе работы должны появиться все элементы, но соблюдение указанного порядка необязательно. Конечно, если бы атрибут `occurs`

одного из элементов имел значение "ZEROORMORE", вызов этого элемента стал бы необязательным.

Следующий код указывает, что должен появиться только один из элементов, конечно, с учетом требований, налагаемых на элементы их собственными атрибутами occurs.

```
<elementType id= "book">
   <group groupOrder="OR">
       <element type="#title" occurs="REQUIRED"/>
       <element type="#author" occurs="ONEORMORE"/>
       <element type="#isbn"/>
       <element type="#publication-date"/>
       <element type="#book-type"/>
       <element type="#book-family" occurs="ONEORMORE"/>
   </group>
</elementType>
```

Содержание, отличающееся от элементного содержания

В нашем примере все остальные элементы имеют в качестве содержания только привычное PCDATA. Как можно описать его, используя элемент string? Очень просто:

```
<elementType id= "title">
   <string/>
</elementType>
<elementType id= "author">
   <string/>
</elementType>
<elementType id= "isbn">
   <string/>
</elementType>
<elementType id= "publication-date">
   <string/>
</elementType>
<elementType id= "book-type">
   <string/>
</elementType>
<elementType id= "book-family">
   <string/>
</elementType>
```

Другие разрешенные ключевые слова (ANY и EMPTY) употребляются для смешанных (mixed) данных. В классическом DTD смешанные данные описываются так:

```
<!ELEMENT mega-element (#PCDATA|[list of allowed elements])*>
```

В XML-данных использование смешанных данных сходно с их использованием в классическом DTD. Описываются они следующим образом:

```
<elementType id= "mega-element">
   <mixed>
       <element type= "#a">
```

```
        <element type= "#b">
        <element type= "#etc">
    </mixed>
</elementType>
```

где дочерние элементы были уже описаны как:

```
<elementType id="#a">
    </string>
</elementType>
etc...
```

Ключевое слово `ANY` используется следующим образом:

```
<elementType id= "mega-element2">
    <any/>
</elementType>
```

Это означает, что элемент `mega-element2` может содержать любое количество элементов, но *не может* содержать строки.

А вот описание для пустого элемента:

```
<elementType id= "mini-element">
    <empty/>
</elementType>
```

Хотя было бы достаточно просто обозначить `elementType` как пустой (`empty`) элемент.

```
<elementType id= "mini-element"/>
```

Определение атрибутов в XML-данных

В классическом DTD атрибуты определяются в описании `ATTLIST`, а именно:

```
<!ATTLIST book
    id ID #REQUIRED>
<!ATTLIST author
    rank CDATA #IMPLIED>
```

В XML-данных для определения атрибута используется элемент `attribute`. Элемент `attribute` обладает своими собственными атрибутами (табл. 3.5).

Таблица 3.5. Атрибуты элемента attribute

atttype (тип атрибута)	Определяет тип атрибута. По умолчанию имеет значение CDATA, то есть нет необходимости включать атрибут atttype, если значение атрибута имеет тип CDATA
values (значения)	Должен быть включен тогда и только тогда, если атрибут attype принимает значение ENUMERATION (перечисление) или NOTATION (обозначение) Смотрите пример в следующем разделе
default (значение по умолчанию)	Значение по умолчанию может быть определено для атрибута типа ENUMERATION. Оно должно быть одним из указанных значений

В нашем примере элемент `book` принимает вид:

```
<elementType id= "book">
```

```
    <element type="#title" occurs="REQUIRED"/>
    <element type="#author" occurs="ONEORMORE"/>
    <element type="#isbn"/>
    <element type="#publication-date"/>
    <element type="#book-type"/>
    <element type="#book-family" occurs="ONEORMORE"/>

    <!-- Здесь определяются атрибуты книги. -->

    <attribute name="id" atttype="ID">

</elementType>
```

И атрибут для элемента `author` появляется следующим образом:

```
<element type="#author" occurs="ONEORMORE"/>
    <!-- Атрибут author определяется здесь -->
    <attribute name="rank" atttype="CDATA"/>
```

Включать атрибут `atttype` для элемента `author` нет необходимости, поскольку `CDATA` само является значением по умолчанию.

Другие значения, принимаемые атрибутом `atttype`, соответствуют значениям, встречающимся в классическом DTD, а именно:

```
(URIREF | ID | IDREF | IDREFS | ENTITY | ENTITIES | NMTOKEN | NMTOKENS |
ENUMERATION | NOTATION | CDATA)
```

По умолчанию используется значение `CDATA`. Подробности можно увидеть в спецификации, находящейся на сайте:

http://www.w3c.org/TR/1998/NOTE-XML-data/

Пример перечисления

В нашем примере до сих пор ни разу не встречался атрибут с перечисляемыми значениями. Запись, приведенная ниже, взята из третьей главы книги Professional Style Sheets, Wrox Press, ISBN 1-861001-65-7. Напомним, что атрибутом с перечисляемыми значениями называется атрибут, который предоставляет список значений, а также значение по умолчанию.

```
<!ATTLIST attitude
   interest(hot|warm|cool|unknown) "unknown">
```

В классическом DTD не обязательно сообщать о перечисляемых значениях, синтаксис делает это очевидным. В XML-данных приходится определить необходимый тип, используя атрибут `atttype`.

Пример, показанный выше, будет воспроизведен в XML-данных следующим образом:

```
<elementType id="attitude">

   <!-- Здесь появляются типы элементов. -->

   <!-- Здесь определяются атрибуты отношений с перечисляемыми значениями..
   ..и по умолчанию -->
   <attribute name="interest" atttype="ENUMERATION" values= "hot warm cool
_unknown" default= "unknown">
```

```
</elementType>
```

Мы практически исчерпали тему воспроизведения классического определения типа документа с помощью XML-данных. Учтите, теперь определение типа документа само является XML-файлом, к которому следует с помощью XML-анализатора применить критерий состоятельности. Ниже приводится законченная форма нашего примера.

```
<?xml version= "1.0" ?>
<schema xmlns="urn:W3C.org:xmlschema"
   xmlns:dt="urn:W3C.org:xmldatatypes">
<elementType id="wrox-books-table">
   <element type="#book" occurs="ONEORMORE"/>
</elementType>

<elementType id= "book">
   <element type="#title" occurs="REQUIRED"/>
   <element type="#author" occurs="ONEORMORE"/>
       <!-- Здесь определяется атрибут author -->
       <attribute name="rank" atttype="CDATA"/>
   <element type="#isbn"/>
   <element type="#publication-date"/>
   <element type="#book-type"/>
   <element type="#book-family" occurs="ONEORMORE"/>

   <!-- Здесь определяется атрибут book -->
   <attribute name="id" atttype="ID">

</elementType>

<elementType id= "title">
   <string/>
</elementType>

<elementType id= "author">
   <string/>
</elementType>

<elementType id= "isbn">
   <string/>
</elementType>

<elementType id= "publication-date">
   <string/>
</elementType>

<elementType id= "book-type">
   <string/>
</elementType>

<elementType id= "book-family">
   <string/>
</elementType>

</schema>
```

Теперь можно перейти к вопросу, как XML-данные расширяют возможности DTD. У этой проблемы есть две стороны.

Открытые модели содержания

Если в XML-документ с определением типа документа добавить еще один элемент, называющийся `<author-biog>`.

```
<wrox-books-table>

   <book id="b1">
       <title></title>
       <author rank=""></author>
<author-biog></author-biog>
       <isbn></isbn>
       <publication-date></publication-date>
       <book-type></book-type>
       <book-family></book-family>

   </book>

</wrox-books-table>
```

то браузер воспримет эту вставку как ошибку. Однако в XML-данных можно использовать другой атрибут – `content`, чтобы разрешить введение типов элементов, которые не были описаны явно, и сохранить при этом состоятельность документа. Это осуществляется с помощью моделей содержания OPEN (открытая) или CLOSED (закрытая).

```
<elementType id="book" content="CLOSED">
```

Если значение атрибута `content` равно `"CLOSED"`, то совместимый с XML-данными браузер будет по-прежнему воспринимать введение нового элемента как ошибку. Но если значение `content` станет `"OPEN"`:

```
<elementType id="book" content="OPEN">
```

или вообще никаким (то есть значение `"OPEN"` присваивается по умолчанию), то браузер, не возражая, воспримет неописанный элемент.

Невозможно переоценить важность этого свойства XML-данных. В случае классического DTD при добавления в программу хотя бы одного-единственного элемента необходимо полностью модифицировать DTD. Если используются XML-данные, у программиста появляется выбор между строго навязанной схемой (учтите, что при работе с документами строгость имеет ряд серьезных преимуществ) и более свободным отношением к включению в описание тех или иных новых элементов. При этом следует иметь в виду: что более свободное отношение к DTD означает, что документ, если он содержит неописанные элементы, подобные приведенным выше, может оказаться только правильным, поскольку новые элементы нарушат его состоятельность. В то же время, использование такого подхода очень важно, если по ходу дела приходится строить таблицы из базы данных.

Содержание элемента по умолчанию

Если значение атрибута `occurs` для какого-либо элемента равно `"REQUIRED"` или

"OPTIONAL", другими словами, этот элемент может встречаться в группе только один раз, то для его содержания можно определить значение по умолчанию <default> в качестве содержания элемента.

В примере, приведенном выше, можно было бы определить значение по умолчанию "О компьютерах" в качестве содержания элемента <book-type>.

Это можно сделать следующим образом:

```
<element type="book-type"occurs="REQUIRED" >
   <default> О компьютерах </default>
</element>
```

Теперь, если никакое содержание для элемента не предоставляется, браузер подставит значение "О компьютерах".

Для того чтобы разрешить вставлять только такое содержание, можно использовать пару атрибут-значение presence="FIXED" следующим образом:

```
<element type="#book-type" occurs= "REQUIRED" presence = "FIXED">
   <default> О компьютерах </default>
       </element>
```

В результате совместимый с XML-данными анализатор при попытке добавить элементу book-type любое содержание, кроме содержания "О компьютерах", будет каждый раз выдавать ошибку.

Свойства XML-данных

Кроме выше рассмотренных достоинств, XML-данные обладают дополнительными возможностями, с помощью которых можно не только расширить функциональность DTD, но и добиться определенных примуществ при использовании XML для создания базы данных или хранения объектов.

Псевдонимы и корреляты

Для двуязычных документов обычным явлением считается, когда различные слова означают одно и то же. Например, руководство по просьбе французского отделения издательства Wrox поставило задачу изменить в каталоге наши тэги на французские. Для этой цели вполне подойдет элемент схемы, называемый elementTypeEquivalent. Эту операцию можно проделать следующим образом:

```
<elementTypeEquivalent id= "livre" type= "book"/>
<elementTypeEquivalent id= "auteur" type= "author"/>
```

Другим примером могут служить термины, с помощью которых описывается один и тот же объект. Например, в базе данных Wrox в элементе wrox-books-table могут быть такие сведения:

```
<book>
   <title>Professional style Sheets for HTML and XML</title>
   <author>Frank Boumphrey</author>
...etc.
</book>
```

Впоследствии, при расширении базы данных, возникла следующая запись:

```
<person>
   Frank Boumphrey
   <works> Professional style Sheets for HTML and XML</works>
...etc.
</person>
```

Как объяснить анализатору, что `<title>` и `<works>` описывают одинаковые объекты? Для этого необходимо следующим образом использовать элемент схемы `<correlative>`:

```
<elementType id="title">
   <string/>
</elementType>

<elementType id= "works">
   <correlative type= "#title"/>
   <string/>
</elementType>
```

Обратите внимание на синтаксис: мы обязаны вставить значок `#` перед именем элемента, с которым намерены коррелировать наш элемент.

Иерархии классов

Типы элементов могут быть объединены в группы при помощи элемента `superType`.

В следующем примере элемент `superType` указывает на то, что как элемент `last-name`, так и элемент `first-name` относятся к типу `name`.

```
<elementType id= "author">
   <string/>
</elementType>

<elementType id= "name">
   <element type= "#author"/>
</elementType>

<elementType id= "last-name">
   <superType type= "#name"/>
   <element type= "#author"/>
</elementType>

<elementType id= "first-name">
   <superType type= "#name"/>
   <element type= "#author"/>
</elementType>
```

Таким образом, элементы `last-name` и `first-name` являются подмножествами типа `name`. Заметим, что тип `name` должен иметь открытую модель содержания: `content="OPEN"`.

Ключевые ссылки

Используя XML-данные, мы можем придумывать элементы, играющие роль ссылок на другие ссылки, для чего необходимо использовать так называемые

ключевые ссылки (key references). Вернемся к нашему примеру с книгами, в котором приводились объекты, названия которых необходимо было выразить двумя различными способами:

```
<book>
       <title>Professional style sheets for HTML and XML</title>
       <author> Frank Boumphrey</author>
</book>

<!-- ..и позже, в том же самом документе.. -->

<person>
   <name>Frank Boumphrey</name>
       <works>Professional style sheets for HTML and XML</works>
</person>
```

Строка `Frank Boumphray` является значением элемента `<author>`, дочернего к элементу `<book>`, а также элемента `<name>`, дочернего к элементу `<person>`. Мы можем выразить это соотношение с помощью элементов `key`, `keyPart` и `ForeignKey`.

Сначала опишем элемент:

```
<elementType id="name">
   <string/>
</elementType>
```

Затем сообщим, что элемент `name` является дочерним к элементу `person`, и присвоим ему значение `id`, равное `a_person1`. Используя элемент `key`, создадим значение `id`, равное `a_key1`, которое объявляет, что личность человека (`person`) может быть единственным образом определена его именем (`name`), а при помощи элемента `keyPart` объявим, что оно связано с элементом, значение `id` которого равно `a_person1`, то есть с элементом `name`.

```
<elementType id="person">
   <element id="a_person1" type="#name"/>
   <key id="a_key1"><keyPart href="#a_person1"/></key>
   <string/>
</elementType>
```

Следующая запись информирует представителя пользователя о том, что содержание элемента `author` (автор) представляет собой строку с ключом (`foreignKey`), идентифицирующим автора по имени (`name`).

```
<elementType id="author">
   <string/>
   <foreignKey range="#person" key="#a_key1"/>
</elementType>
```

Эти описания могут показаться довольно громоздкими, а сам замысел достаточно сложным для понимания, однако при обработке баз данных они могут понадобиться.

Для дальнейшего обсуждения соотношений типа «один-ко-многим», использующих атрибуты в качестве ссылок и многокомпонентные ключи (multipart

keys), лучше обратиться к спецификации. Соотношения такого типа можно будет пользоваться только в том случае, когда приводимые здесь методики станут общепризнанными и закрепленными стандартом, однако знакомство с ними проясняет перспективу описания взаимоотношений между элементами и объектами.

Ограничения

Еще одним важным и очень полезным достоинством XML-данных является концепция ограничений. Она позволяет «ограничивать» содержание, которое помещается в элемент. Например, нам может понадобиться поставить рамки для суммы денег, снимаемых с банковского счета. Для этого мы используем элементы `<max>` и `<min>`.

Очевидно, нельзя разрешить, чтобы взятая из банка сумма была отрицательной (это было бы то же самое, что давать кредит по счету!), а максимальную снимаемую сумму можно установить в 500 долларов. Далее показано, как это делается.

```
<elementType id= "withdrawal">
   </string>
</elementType>

<elementType id= "account">
   <element href= "#withdrawal"><min>0</min><max>500</max>
</elementType>
```

Снимаемая со счета сумма должна быть преобразована в отличную от строки форму, что подводит нас к еще одному существенному преимуществу схем, основанных на XML – возможности определять типы данных. Вот как это предлагается делать.

Типы данных

Проблема классического DTD состоит в том, что содержание элемента в нем определяется либо как другой элемент, либо как данные типа `PCDATA`, которые представляют собой законченную строку. Этот метод, однако, не дает возможности сообщать пользователю, что содержание элемента является датой, целым числом со знаком или без знака, вещественным числом, логической константой или URL. Очевидно, что такая вариативность порой бывает просто необходима, особенно в тех случаях, когда требуется воспользоваться элементами либо для определения объектов, либо как частью базы данных, которая впоследствии должна тем или иным способом интерпретироваться.

Возвращаясь к нашему классу `hat`, запишем:

```
<hat id="h2">
   <style>Stetson</style>
   <size>7</size>
   <color>gray</color>
</hat>
```

В чем программисты действительно испытывают нужду, так это в инструменте, который давал бы возможность определять природу содержания дочерних элементов или их свойств. Например:

```
<hat id="h2">
   <style dataType= "string">Stetson</style>
   <size dataType= "integer">7</size>
   <color dataType= "string">gray</color>
</hat>
```

Конечно, можно ввести такой атрибут и в классическое DTD, и он бы описывал, как нужно интерпретировать данные, но содержание все равно оставалось бы строкой, и анализатор не выдавал бы ошибку при вводе следующей информации: `<size dataType="integer">large</size>`. А ведь очевидно, что `large` не является целым числом!

Можно обойти эту трудность, описывая пространство имен для типов данных, как в нашем первом примере (здесь он, правда, немного подправлен). Напомним:

```
<?xml version= "1.0" ?>
<schema xmlns="urn:W3C.org:xmlschema"
   xmlns:dt="urn:W3C.org:xmldatatypes">

   <elementType id= "xdoc">
       <element type="#greeting"/>
       <element type="# days-taken-for-creation "/>
   </elementType>

   <elementType id= "greeting">
       <string/>
   </elementType>

   <elementType id= "days-taken-for-creation">
       <int/>
   </elementType>
</schema>
<xdoc>
   <greeting> Hello World! </greeting>
   <days-taken-for-creation dt:dt= "int">7</days-taken-for-creation>
<xdoc>
```

Как агент пользователя справится с подобной информацией, это его дело. Ему, по крайней мере, сообщили: «Содержание элемента `days-taken-for-creation` должно быть целым числом. Будьте добры, обеспечьте это условие». Типы данных, которые распознаются как XML-данные, так и DCD, приведены в «Приложении D».

Мы более или менее завершили обзор XML-данных. В нем, к сожалению, пришлось обойти некоторые менее ясные стороны рассматриваемого вопроса, однако, как нам кажется, на предыдущих страницах представлено достаточно материала, чтобы дать общую картину достоинств этой схемы по сравнению с классическим DTD. Конечно, пока еще не существует приложений, поддерживающих XML-данные, и вряд ли они появятся раньше, чем через год-другой, однако браузер IE5 и в настоящее время поддерживает пространства имен для типов данных. Другими словами, с его помощью вы можете отметить, что содержание относится к опре-

деленному типу данных. Например:

```
<price dt:dt="fixed.14.4">1.95</price>
```

Можно быть уверенным, что как только будет принято решение о какой-то стандартной схеме, приложения будут ее поддерживать. Теперь обратимся к другой предложенной методике, которую мы упоминали ранее – DCD для XML.

Описание содержания документа

Предложение по этому вопросу было представлено в консорциум W3C в июле 1998 года. Как и в случае XML-данных, оно пока находится на стадии сообщения. Описание содержания документа (DCD) использует модифицированный синтаксис формата описания ресурсов (Resource Description Framework, RDF) и включает подмножество предложения по XML-данным в виде, совместимом с RDF. Напоминаем, что мы кратко обсуждали RDF и его возможности во «Введении».

DCD все определеннее идет по пути описания *ограничений* (constraints), налагаемых на структуру и содержание XML-документов, однако рассматривать этот вопрос с той же полнотой, как XML-данные, мы не станем. На этих страницах будет дан общий обзор DCD, который обеспечит читателей достаточно полной информации; с ее помощью они смогут сами разобраться во всех достоинствах этого метода и открываемых им перспективах. Сообщение по DCD удивительно хорошо написано, легко читается, его можно загрузить и получить более подробную информацию о синтаксисе. Для этого обратитесь по адресу:

http://www.w3c.org/TR/Note-dcd

Спецификация построена в соответствии с приводимыми сдесь принципами, цитируемыми по упомянутому описанию.

Цитата

1.2 Принципы построения.

1. *Семантика DCD должна в качестве подмножества включать в себя семантику определения типа документа XML.*
2. *Модель данных и синтаксис DCD должны соответствовать модели данных и синтаксису RDF.*
3. *Ограничения в DCD должны позволить создающим документы программам, а также другими приложениями, нуждающимися в необходимой информации о структуре и содержании документа, использовать его непосредственно.*
4. *DCD должно иметь возможность использовать механизмы других рабочих групп консорциума W3C там, где эти механизмы уместны и целесообразны.*
5. *DCD должно быть читаемо человеком, и при этом быть достаточно понятным.*

Обратите внимание, чтобы DCD могло удовлетворять второму принципу построения, требуется определенная модификация синтаксиса RDF.

Цитата

2.1.1 Предлагаемое упрощение синтаксиса RDF.

Как уже было сказано, предполагается, что DCD должно соответствовать модели и синтаксису спецификации RDF. Однако это условие требует определенных упрощений в синтаксисе RDF, о которых мы собираемся сообщить рабочей группе RDF. Этот синтаксис будет принят только после его утверждения рабочей группой RDF. Синтаксические упрощения состоят в следующем:

RDF:li

RDF:li не должно вызываться, если в совокупность вставляются типизированные узлы.

Collection type (Тип совокупности).

Тип совокупности свойств может быть определен как атрибут узла.

Дополнительную информацию об RDF можно найти по адресу:

http://www.w3c.org/RDF/Overview.html

На этой же странице есть ссылка на синтаксис RDF, описание которого находится по адресу:

http://www.w3c.org/TR/WD-rdf-syntax/

Далее будет дан краткий обзор DCD. Заметим, что принципы его построения такие же, как и у XML-данных. Другими словами, если до сих пор вы не испытывали трудностей с пониманием излагаемого материала, вам удастся разобраться и с DCD. Дальше будет показано, что в большинстве случаев DCD представляет собой оптимизацию и упрощение схемы XML-данных.

Чтобы подчеркнуть синтаксические различия между первым и вторым приемом, вспомним об упоминавшемся уже «Здравствуй, мир!», только на этот раз вместо XML-данных применим к выражению DCD.

Описание содержания документа – начнем с простого

Прежде всего стоит еще раз взглянуть на то, каким образом «`Здравствуй, мир!`", был отображен в синтаксисе XML-данных:

```
<?xml version= "1.0" ?>
<schema xmlns="urn:W3C.org:xmlschema"
   xmlns:dt="urn:W3C.org:xmldatatypes">
<elementType id= "greeting">
   <string/>
</elementType>
```

```
</schema>

<greeting> Hello World! </greeting>
```

А теперь приведем один из способов, которым он может быть воспроизведен в DCD:

```
<DCD>
   <ElementDef>
       <type>greeting</type>
       <model>Data</model>
   </ElementDef>
</DCD>

<greeting> Hello World! </greeting>
```

В этом примере свойства элемента `greeting` отражены как дочерние элементы элемента `ElementDef`, что представляет собой сокращение от Element Definition (определение элемента).

Взаимозаменяемость элементов и атрибутов

Одна из сильных сторон предложения по DCD заключается в том, что при использовании синтаксиса RDF неповторяющиеся свойства могут быть выражены как атрибуты родительского элемента, а не как дочерние элементы. Именно это мы и продемонстрируем в приведенном ниже примере. Здесь `Type` и `Model` могут быть использованы или как элементы или как атрибуты (но, разумеется, не то и другое одновременно):

```
<DCD>
   <ElementDef Type="greeting" Model="Data">
       <Description>Our first simple example</Description>
   </ElementDef>
</DCD>
<greeting> Hello World! </greeting>
```

Заметим, что наш элемент `greeting` определяется в DCD элементом `ElementDef` (аналогичным элементу `elementType` в XML-данных), и его атрибутом `Type` (аналогичным атрибуту `id` в XML-данных). Свойство `Model` элемента `ElementDef` принимает значение `Data`, указывая на то, что элемент содержит единственное значение данных. Тип данных по умолчанию эквивалентен типу `<string/>` в XML-данных.

Обращаем также ваше внимание на использование элемента `Description` во втором примере. Он может показаться похожим на комментарий – конечно, вы имеете право использовать комментарии XML, однако элемент `Description` позволяет описывать в разметке семантику, с помощью которой приложения смогут поддерживать затем действия пользователя.

Совет *Не забывайте, что в языке XML заглавные и строчные буквы различаются, так что обязательно проверьте, правильно ли вы их расставили!*

Ограничение синтаксиса

Прежде чем продолжить обсуждение DCD, следует объяснить, что представляет собой необязательная команда приложения (PI), определенная в начале ее спецификации, поскольку такое дополнение вполне может вызвать путаницу. В разделе 2.1.2 спецификации DCD вводится так называемая необязательная команда PI:

```
<?DCD syntax="explicit"?>
```

Когда строго необязательная PI включена, она должна находиться сразу после открывающего тэга `<DCD>`. Подобная вставка означает, что указанные ниже свойства *должны* быть определены с использованием синтаксиса атрибутов:

`Type` (тип), `Model` (модель), `Occurs` (сколько раз встречается),
`RDF:Order` (порядок следования), `Content` (содержание),
`Root` (корень), `Fixed` (фиксированный), `Datatype` (тип данных).

Скоро мы вновь встретимся с этими свойствами, но вот о чем умалчивает спецификация, – почему команда PI находится именно после открывающего тэга? В период создания DCD некоторые разработчики размышляли примерно так: возможность определять свойства либо как атрибуты, либо как элементы чрезмерно обобщена и приведет к тому, что анализаторы будут иметь большие размеры и станут медленно работать. Команда PI включается для того, чтобы ограничить гибкость синтаксиса. Подобное ограничение должно позволить создавать небольшие и быстрые анализаторы. Зная, что такая команда строго необязательна (и представляя себе, зачем нужно ее использовать), вы сможете легко разобраться в остальной части спецификации.

Узлы DCD и типы ресурсов

В пространстве имен DCD определено несколько типов узлов или ключевых элементов. Заметим, что увидеть в браузере соответствующее пространство имен невозможно, так как его содержание описано исключительно в спецификации. Кроме того, это пространство не нужно включать в документ, поскольку оно подразумевается в тэге `<DCD>`.

К узлам и ключевым элементам относятся: `DCD ElementDef` (определение элемента), `Group` (группа), `AttributeDef` (определение атрибута), `ExternalEntityDef` (определение внешнего объекта), `InternalEntityDef` (определение внутреннего объекта).

Совет *Еще раз напоминаем, что в языке XML заглавные и строчные буквы различаются, так что обязательно удостоверьтесь, что вы правильно их расставили!*

Как элементы и атрибуты рассматриваются в DCD

Помимо описания, какие типы элементов, атрибутов и значений могут быть применены в данном документе, в спецификации подчеркивается, что DCD ог-

раничивает также структуру и содержание документа, который его использует. Это может показаться лишним, поскольку то же самое могут делать все схемы, однако такой подход лежит в основе концепции применения данной методики. Существует также возможность ограничения документов с помощью более чем одного DCD.

Как было видно из приведенного выше примера, типы элементов описываются с использованием элемента `ElementDef`. Затем на эти элементы путем присвоения им свойств накладываются специальные ограничения. Определения элементов указывают на то, что они могут иметь дочерние элементы, а также атрибуты с теми или иными именами и свойствами. Дочерние элементы должны быть собраны в группы (`Groups`) там, где они определяются. Группы требуют наличия свойств `Order` (порядок следования) и `Occurs` (сколько раз встречается). Каждый элемент `ElementDef` должен иметь свойство `Type`, которое служит той же цели, что и `id` в XML-данных, то есть дает имя типу элемента. Конечно, чтобы соответствовать требованиям состоятельности, свойство `Type` должно быть уникальным внутри DCD. Как только читатели познакомятся с понятием *группы элементов* подробнее, в тексте сразу будет приведен пример, а до того внимание следует обратить на то, что спецификация DCD позволяет также вводить атрибуты и элементы из других DCD. На них ссылаются с помощью пространств имен (которые будут детально рассмотрены в следующей главе). Коротко отметим, что пространство имен XML представляет собой совокупность имен, идентифицируемых по какому-либо URI и используемых в XML-документах в качестве типов элементов и имен атрибутов. Если элементы принадлежат одному и тому же DCD, на них ссылаются по их свойству `Type`. Если же они используют эти пространства, чтобы сослаться на другое DCD, `Type` может быть условным именем, в котором префикс указывает на пространство имен.

Атрибуты описываются с использованием элемента `AttributeDef`. Они могут быть использованы сами по себе или внутри определений элементов. Каждый атрибут должен иметь свойство `Name`, которому в свою очередь приписывается свойство `Global` (глобальный). Его значение является булевой константой. Эта величина указывает, должно ли имя атрибута быть уникальным для данного DCD или нет. Если значение `Global` равно `True`, то свойство `Name` должно быть уникальным внутри DCD. Если же значение `Global` определено как `False`, то другие элементы могут иметь атрибуты с тем же именем.

Глобальный атрибут можно применять в любом определении элемента внутри DCD, просто указав его имя. В то же время на глобальные атрибуты из других пространств имен следует ссылаться, используя условные имена. (Условные имена будут подробнее рассмотрены в главе 4, как раз посвященной этому вопросу.) Локальные атрибуты, для которых значение свойства `Global` равно `False`, могут быть использованы только внутри того элемента `ElementDef`, где они описаны.

В нашем примере мы уже видели элементы `DCD` и `ElementDef` в действии, и прежде чем перейти к примеру с базой данных издательства Wrox, давайте кратко рассмотрим эти и другие элементы.

Элемент DCD

Элемент `DCD` служит контейнером для описания содержания документа. Типы элементов, которые имеют право появляться внутри него, должны соответствовать типам свойств RDF. Вот свойства, которые может иметь `DCD`: `ElementDef` (определение элемента), `AttributeDef` (определение атрибута); `Description` (описание), `InternalEntityDef` (определение внутреннего объекта); `ExternalEntityDef` (определение внешнего объекта), `Content` (содержание); `Namespace` (пространство имен).

Свойство AttributeDef

Как мы уже видели, атрибуты могут определяться в качестве свойств DCD. Далее будет показано, что атрибуты также могут быть определены внутри элемента `ElementDef`.

Свойство Description

Это свойство используется в том случае, когда необходимо создавать доступные для чтения человеком описания семантики и использования DCD.

Свойства ExternalEntityDef и InternalEntityDef

Мы вернемся к ним позже. Заметим только, что они указывают на объекты, которые могут быть вызваны путем ссылок на них. Эти описания совпадают с описаниями DTD для указанных типов объектов.

Свойство Content

Свойство `Content`, если его значение равно `Open` (открыто), разрешает появление в документе элементов, которые не были определены в элементе `ElementDef`. Использование подобных элементов будет означать, что документ не считается состоятельным, пока анализатор не обработает внешние объекты, которые могут содержать иные определения объектов, где и были указаны данные элементы. Другим значением, которое может принимать свойство `Content`, является `Closed` (закрытое). Оно означает, что все использованные элементы должны быть описаны в элементе `ElementDef`.

Свойство Namespace

Это свойство обязательно для каждого DCD и открывает для него пространство имен. Его значением должен быть URN, который как раз и идентифицирует подобное пространство. Поскольку данное свойство применяется ко всем элементам и атрибутам, присоединяемым к DCD посредством свойств, оно позволяет ссылаться на элементы и атрибуты конкретного документа из более, чем одного DCD. Еще раз повторяем, этот вопрос будет подробно рассмотрен в следующей главе.

Элемент ElementDef

`ElementDef` представляет собой описание типа элемента. Ниже указаны свойства, которые может приобретать элемент `ElementDef`, либо в виде дочерних элементов, либо в виде атрибутов: `Type` (тип), `Model` (модель), `Attribute` (атрибут), `AttributeDef` (определение атрибута), `Group` (группа), `RDF:Order` (порядок следования), `Content` (содержание), `Description` (описание), `Root` (корень), `Default` (по

умолчанию), Fixed (фиксированный), Min (минимум), Max (максимум), Datatype (тип данных).

Свойство Type

Каждый элемент ElementDef в DCD должен иметь свойство Type, которое описывает элемент, определяемый с помощью ElementDef. Когда свойство Type используется в сочетании с элементом ElementDef, его значение не может начинаться с префикса или двоеточия.

Уточнение *Это условие необходимо потому, что существует вероятность ссылки на элементы и атрибуты из другого DCD с помощью пространств имен. Определения элементов, находящихся в другом DCD, могут иметь условное имя как значение свойства Type, префикс которого указывает на пространство имен. Если префикс или двоеточие используются в качестве значения Type в определениях внутренних элементов, это может потребовать от анализатора искать несуществующее пространство, что вызовет ошибку.*

Свойство Model

Определяет тип содержания, которое может иметь элемент. Свойство Model принимает значения, указанные в табл. 3.5.

Таблица 3.6. Возможные значения свойства Model

Значение	Содержание
Empty (пустое)	Ничего не содержит
Any (любое)	Любое содержание, удовлетворяющее требованиям XML
Data (данные)	Одиночное значение данных
Elements (элементы)	Только дочерние элементы, при этом тип элемента контролируется свойствами Group и Element
Mixed (смешанное)	Текст и встроенные дочерние элементы, при этом тип элемента контролируется свойством Element

По умолчанию используется содержание Data.

Уточнение *Заметим, что определение, которое дается в DCD для модели Any, такое же, как и для модели содержания ANY в классическом DTD. А вот определение XML-данных для модели Elements ограничивает содержание только элементами и по непонятным причинам исключает текст!*

Свойство AttributeDef

Свойство AttributeDef может также появляться как свойство элемента ElementDef.

Свойство Attribute

Это свойство определяет атрибуты, которые может принимать элемент. Свойс-

тва Attribute обязаны быть уникальны для каждого определения элемента. То есть элемент <greeting> не может, например, иметь два атрибута, которые оба называются friendly. Однако различные определения элемента могут использовать то же самое (глобальное) значение свойства Attribute. То есть как элемент <greeting>, так и элемент <salutations> могут иметь атрибут friendly.

Свойства Group, Occurs и Order

Если модель содержания элемента ElementDef принимает значение "Elements", то ElementDef должен иметь свойство Group. Это свойство определяет элементы и группы, которые могут встретиться как дочерние для элементов типа, определенного элементом ElementDef.

В свою очередь, свойство Group может иметь свойство Occurs, для которого допустимы значения Required (обязательный), Optional (необязательный), OneOrMore (один или более), ZeroOrMore (ноль или более). По умолчанию используется значение Required.

Порядком следования дочерних элементов управляет атрибут RDF:collection. Разрешенными значениями являются Seq, при котором элементы должны появляться последовательно, или Alt, когда может появляться только один из определенных дочерних элементов. Возможно, в дальнейшем будет разрешено значение Bag, которое позволит дочерним элементам появляться в любом порядке. Вот пример, взятый из предложения по DCD:

```
<ElementDef Type="employee" Model="Elements" Content="Closed">

   <AttributeDef Name="employment" Occurs="Required" Datatype="enumeration">
       <Values>Temporary Permanent Retired</Values>
   </AttributeDef>

<Group RDF:Order="Seq">

   <Element>FirstName<?element>
   <Group Occurs="Optional"><Element>MI</Element></Group>

   <Element>LastName<?Element>
   <Group Occurs="OneOrMore" RDF:Order="Alt">
   <Element>Street</Element><Element>PO-Box</Element>
   </Group>

   <Group RDF:Order="Seq">

   <Element>Telephone</Element>
   <Element>Salary</Element>

   </Group>

</Group>
</ElementDef>
```

Данный пример предполагает совместное использование описания содержания документа, а также модели RDF в совокупности со спецификацией синтаксиса RDF. Однако требование совместимости, как уже было сказано ранее, обязывает произвести определенные упрощения синтаксиса RDF, которые авторы спецификации DCD предлагают сделать рабочей группе RDF.

Свойство Content

Оно описывает, может ли данный тип элемента включать дочерние элементы, которые не определены явно в его свойстве `Group`. Принимает значение `Open` или `Closed`. Значение `Open` разрешает включать те типы элементов, которые не были включены в элемент `Group`. Значение `Closed` позволяет включать только те дочерние элементы, которые были описаны в элементе `Group`. Значение по умолчанию выпало из спецификации, однако авторы утверждают, что оно равно `Closed`.

Свойство Root

Это свойство описывает, может ли тот тип элемента, с которым оно применяется, быть использован в качестве корневого элемента соответствующего документа. Свойство `Root` принимает булевы значения, по умолчанию оно равно `False`. Если никакой элемент не имеет этого свойства, тогда любой элемент может быть корневым. Если несколько элементов имеют свойство `Root` со значением `True`, тогда любой из этих типов элементов может быть корневым элементом соответствующего документа.

Свойство Default

Предоставляет значение по умолчанию, как и в XML-данных, например:

```
<ElementDef Type="AirTicketClass">
   <Default>Y</Default>
</ElementDef>
```

Свойство Fixed

Определяет, что значение по умолчанию является единственным разрешенным. Свойство `Fixed` может принимать значение `True` или `False`, например:

```
<ElementDef Type="Namespace" Model= "Data" Fixed="True">
   <Default>http://www.wrox.com/namedefs</Default>
</ElementDef>
```

Свойства Min и Max

Ограничивают разрешенные значения типов элементов. Если никакой тип данных не определен, то значения `Min` и `Max` рассматриваются как строки, а ограничение сверху и снизу производится в процессе проверки на состоятельность.

Свойство Datatype

Типы данных очень похожи на типы данных в XML-данных (но не совпадают с ними). Обратитесь к подробной информации в спецификациях XML-данных и DCD. По умолчанию используется тип `string`.

Элемент AttributeDef

Неудивительно, что элемент `AttributeDef` определяет атрибуты. Напомним, что он может применяться как свойство элемента DCD или как свойство элемента `ElementDef`. Вот пример, который без сомнения вам знаком:

```
<ElementDef Type="IMG">
   <AttributeDef Name="SR" DataType="URI">
</ElementDef>
```

Элемент AttributeDef может иметь следующие свойства: Name (имя), Global (глобальный), ID-Role (роль идентификатора), Occurs (как часто встречается), Min (минимум), Max (максимум).

Свойство Name

Это свойство требуется для каждого атрибута, оно как раз и дает атрибуту имя. Его значение не может начинаться с префикса или двоеточия, если только префикс или двоеточие не ссылаются на пространство имен.

Свойство Global

Указывает, является ли свойство Name уникальным для данного DCD. Может принимать значения True и False, где True означает, что Name должно быть уникальным. False является значением по умолчанию.

Например, в языке HTML почти все элементы могут иметь атрибуты class, style, name, id, type и так далее. Если бы мы захотели использовать DCD, чтобы определить схему для HTML 4, мы могли бы сделать следующее:

```
<DCD>
<AttributeDef Name="class" DataType="Data" Global="True">
<AttributeDef Name="style" DataType="Data" Global="True">

etc...
```

Это означает, что атрибуты будут присоединяться ко всем элементам ElementDef, описанным в элементе <DCD>, простым добавлением Attribute="class".

Например:

```
<ElementDef Type="DIV" Attribute = "class style etc">
```

Имена атрибутов должны отделяться друг от друга пробелом.

Свойство ID-Role

Используется, чтобы обозначить, что атрибут имеет уникальный идентификатор или уникальную семантику указателя ID. Принимает значения ID, IDREF и IDREFS. Каждое значение соответствует значению в DTD.

Свойство Occurs

Определяет, обязательно ли присутствие атрибута. Принимает значение Required там, где атрибут должен встречаться точно один раз и Optional там, где атрибут может не встречаться или встречаться один раз. Значение по умолчанию – Optional.

Свойства Min и Max

Ограничивают разрешенные значения типа атрибута в точности так же, как они ограничивают разрешенные значения типа элемента, которые мы рассматривали ранее.

Элементы InternalEntityDef и ExternalEntityDef

Эти элементы работают так же, как описания объектов в классическом определении типа документа. Приведем следующий пример:

```
<InternalEntityDef Name="www" Value="World Wide Web"/>
<ExternalEntityDef
Name="boilerplate"SystemId="http://www.legalese.com/boiler.htm"/>
```

Элемент `InternalEntityDef` может иметь свойства `Name` и `Value`, а элемент `ExternalEntityDef` может иметь свойства `Name`, `PublicID` и `SystemID`.

В обоих случаях свойство `Name` предоставляет имя, по которому можно отыскать объект, оно является обязательным. Свойство `Value` содержит текст замены для внутреннего объекта. Свойства `PublicID` и `SystemID` те же самые, что и в классическом определении типа документа.

Типы данных

Типы данных в спецификации DCD основаны на типах, которые поддерживаются языком SQL и другими современными языками программирования. Подробнее мы расскажем о них в «Приложении D».

На этом завершается наш краткий обзор описания содержания документа. Надеемся, читатели согласятся с нами – синтаксис DCD, так же как и XML-данных, прост для изучения. Он имеет многочисленные преимущества и лишен недостатков традиционного определения типа документа.

Заключение

В этой главе мы рассмотрели два предложения по схемам, имеющие в своей основе XML: XML-данные и описание содержания документа (DCD) для XML. Рассматривая их, вы должны были обратить внимание, что обе методики ставят своей целью сделать XML языком, пригодным для обмена информацией между приложениями. При этом мы обсудили следующие вопросы:

- что такое схема и каковы ее функции;
- преимущества и недостатки классического DTD;
- XML-данные – схема, основанная на XML ;
- самое новое предложение по схемам, а именно описание содержания документа (DCD) для XML.

Надеемся, что читатели, познакомившиеся с достоинствами схем, основанных на XML, получили надежный запас знаний, и теперь могут самостоятельно следить за дальнейшими достижениями в области XML-схем.

Глава 4. Пространства имен

Разметка только тогда представляет ценность, когда абсолютно точно известно, что она означает.

Взгляните на этот код:

```
<bzx>14.2</bzx>
```

Без дополнительных пояснений догадаться, о чем здесь идет речь, нельзя. Это может быть и результат в беге на 100 метров с барьерами, и средний возраст половой зрелости, и средняя продолжительность жизни кошек, и состояние Билла Гейтса в миллиардах долларов. (Нет, маловато, скорее это выражение относится к его годовому доходу). Агенту пользователя обязательно нужно знать, к чему относится разметка. Только тогда от нее будет польза!

Теперь рассмотрим другой пример:

```
<hemoglobin>14.2</hemoglobin>
```

Всякий образованный человек может догадаться, что эта запись имеет какое-то отношение к крови, а цифра – к количеству гемоглобина. Любому же медику очевидно, что приведенная строка свидетельствует о концентрации гемоглобина в мг/%.

Данный пример прекрасно иллюстрирует тот факт, что разметка может иметь какое-то, пусть даже неявное, значение для «образованного» пользовательского агента, однако ситуация в целом достаточно сложная. Как было бы хорошо, если бы мы могли упростить ее, другими словами, «научить» любого агента самого разбираться, что означает та или иная разметка!

Следующий пример:

```
<title>Big Sur</title>
```

Все знают, что такое title (заголовок), но что это может означать в выше указанном, конкретном случае – название книги или должность? А может, это титул, почетное звание? (Которое жители поселка Мадвилл, например, не могут даже выговорить.) Если нет дополнительной информации, ответить на этот вопрос трудно.

Было бы просто замечательно, если бы мы могли указать «разумному» пользовательскому агенту, с какими именно заголовками он имеет дело, к чему тот или иной из них относится. Какие возможности открылись бы перед программистами,

если бы при создании документа они были в состоянии взять часть разметки и сообщить агенту: «Удостоверься, что этот раздел взят именно из той книги, название которой приводится». Затем взять другую часть разметки, поместить ее в тот же документ и сказать: «Удостоверься, что эта часть соответствует названию должности».

На самом деле способ, с помощью которого можно достичь поставленной цели, существует. Он подразумевает использование пространств имен.

Мы увидим, что пространства имен способны посодействовать пользователю в двух очень важных случаях. С их помощью можно:

- совмещать документы из двух или более источников, не теряя при этом уверенности, что программа различит, из какого источника взят тот или иной элемент или атрибут;
- по возможности разрешить агенту пользователя доступ к дальнейшему материалу, такому как определение типа документа (DTD) или другому описанию элементов и атрибутов.

О чем говорится в этой главе

Значительная часть этой главы отражает сведения из текущей спецификации пространств имен от 16 сентября 1998 года, которую можно найти по адресу http://www.w3.org/TR/WD-xml-names. В отличие от большинства рабочих проектов консорциума W3C, этот перечень условий представляет собой очень хорошо написанный документ, дополненный прекрасными примерами. Авторы данного сборника советуют читателям как можно скорее познакомиться с ним.

В этой главе будут рассмотрены следующие вопросы:

- что такое пространство имен;
- как их идентифицировать и описывать; каков применяемый для этих целей синтаксис;
- что подразумевается под областью действия в пространствах имен;
- ожидаемое поведение агента пользователя;
- некоторые приложения.

Что такое пространство имен

Пространство имен представляет собой совокупность некоторых величин или характеристик, которые могут быть использованы в XML-документах как имена элементов или атрибутов. Эти характеристики показывают, какой конкретно области знаний они соответствуют. Пространства имен в XML определяются унифицированным идентификатором ресурса (Uniform Resource Identifier, URI). Он позволяет каждому пространству имен быть уникальным, каким оно и должно быть, если требуется искать имя атрибута или элемента по всей сети Internet.

Уточнение *URI является суперклассом, который включает в себя как унифицированный номер ресурса (Uniform Resource Number, URN), так*

и унифицированный локатор ресурса (Uniform Resource Locator, URL). Несмотря на то, что в будущем ожидается возникновение большого количества URN, в настоящее время URI в большинстве случаев означает то же, что и URL. Важно то, что мы имеем уникальный номер или имя, которые могут определить элемент или атрибут универсально и уникальным способом. Почему мы используем URI? Просто потому, что он дает нам число или строку, которые как раз и созданы таким образом, чтобы быть универсальными и уникальными.

Например, может существовать четыре элемента, которые называются `class`. Первый из них относится к учебным классам средней школы в поселке Мадвилл, второй определяет тип мест в самолете, третий – класс химических соединений, и последний – положение английского джентльмена в обществе.

Можно идентифицировать эти классы с помощью соответствующих им различных URI. Первому можно поставить в соответствие URI поселка Мадвилл, второму – URI Федерального управления авиацией, третьему – область химических языков и последнему – URI с книгой пэров. Чтобы связать каждый элемент с соответствующим пространством имен, можно ввести в документ следующие записи:

```
<http://www.mudville-schools.org.class>

<http://www.faa.gov.class>

<http://www.chem-lang.org.class>

<http://www.baronage.co.uk.class>
```

Или:

```
<class namespace="http://www.mudville-schools.org">

<class namespace="http://www.faa.gov/">

<class namespace="http://www.chem-lang.org">

<class namespace="http://www.baronage.co.uk/">
```

и так далее. Но подобная операция связана с рядом проблем, а именно:

- очевидно, что повторение такой процедуры для каждого элемента крайне утомительно;
- неизбежны случаи, когда потребуется использовать два атрибута с одинаковыми именами в одном и том же элементе (а в XML это запрещено);
- идентификаторы URI часто содержат символы, запрещенные для использования в именах XML.

Спецификация пространства имен консорциума W3C предлагает нам гораздо более выгодный способ сопоставления элементов и атрибутов с соответствующими URI. Этим далее мы и займемся.

Идентификация и описание пространств имен

Давайте рассмотрим, каким образом спецификация, утвержденная консорциумом W3C, позволяет нам описывать и идентифицировать пространства имен.

Синтаксис пространств имен

Понятно, что в любом конкретном документе прежде всего необходимо поставить какой-то префикс, соответствующий некоему идентификатору URI, который мы хотим использовать в качестве обозначения пространства имен. Эта операция обычно выполняется с помощью следующего синтаксиса:

```
xmlns:[префикс]="[URI пространства имен]"
```

где `xmlns:` – зарезервированный атрибут. Префикс может состоять из любых символов, разрешенных в тэгах XML, если только они не начинаются с комбинации символов `"xml"` в любой форме (такие строки запрещены). В результате мы объявили префикс в качестве псевдонима для пространства имен.

Теперь, когда возникает задача поставить в соответствие данному локальному элементу или атрибуту конкретное пространство имен, необходимо просто присоединить к нему описанный префикс. Другими словами, если мы описали пространство имен для элемента `class` средней школы поселка Мадвилл следующим образом:

```
xmlns:sch= "http://www.mudville-schools.org"
```

тогда запись:

```
<sch:class>Advanced Basket Weaving</sch:class>
```

сообщает элементу пользователя о том, что мы используем элемент `class`, определенный некоторым способом в некотором месте, как элемент `http://www.mudville-schools.org:class`, но ни в коем случае не как элемент `class` из какой-то другой области.

Совет *Хотя URI используется для определения пространства имен, это еще не означает, что агент пользователя способен найти информацию об использовании данного пространства на указанном сайте. Должен существовать отдельный метод обучения агента. Последнее условие может быть источником серьезных затруднений, возникающих при первом чтении спецификации пространств имен.*

Описание пространств имен

Пространство имен часто описывается внутри корневого элемента документа, но оно также может фиксироваться в любом другом его элементе. Пространство имен не может быть использовано ранее своего описания. Этот вопрос будет подробно изучен, когда мы перейдем к разговору об области действия.

А пока пример для пояснения указанного выше тезиса:

```
<xdoc xmlns:sch="http://www.mudville-schools.org">
```

```
    <sch:class-list>
        <sch:class>Advanced Basket Weaving</sch:class>
        <sch:class>3D Art</sch:class>
        <sch:class>Remedial Reading</sch:class>
    </sch:class-list>
</xdoc>
```

Просматривая этот документ, программисты, как, впрочем, и пользовательский агент, могут сделать вывод, что все элементы с префиксом `sch:`, то есть все элементы внутри этого документа, принадлежат пространству имен `http://www.mudville-schools.org`. Элемент `xdoc`, в котором находится описание, также принадлежит тому же пространству имен, поскольку он содержит атрибут `xmlns:[префикс]`.

Совет *На практике в XML-документе элемент или атрибут, использующий пространство имен, может быть обнаружен по наличию двоеточия в имени тэга. Ключевые слова XML, такие как атрибут xml:space, сообщают как читающему документ человеку, так и агенту пользователя, что атрибут space принадлежит пространству имен "xml" и должен рассматриваться как таковой.*

Теперь представим, что мы хотим добавить элементы `class`, относящиеся к области химии: один для класса соединений, называемых «сложные эфиры», а другой для класса соединений, называемых «бензолы». Тогда исправленный документ будет выглядеть так:

```
<xdoc
    xmlns:sch="http://www.mudville-schools.org"
    xmlns:chem="http://www.chem-lang.org">
    <sch:class-list>
        <sch:class>Advanced Basket Weaving</sch:class>
        <sch:class>3D Art</sch:class>
        <sch:class>Remedial Reading</sch:class>
        <sch:class>
            OrganicChemistryI <chem:class>Esters</chem:class>
        </sch:class>

        <sch:class>
            OrganicChemistryII <chem:class> Benzenes</chem:class>
        </sch:class>
    </sch:class-list>
</xdoc>
```

Теперь и программист, и агент пользователя знают, что второй тип элемента `class`, – тот, который использует префикс `chem`, – принадлежит пространству имен `http://www.chem-lang.org`. Подчеркнем еще раз, что префикс не является пространством имен, это только псевдоним, который выбрал создатель документа, для представления этого пространства.

На самом деле существует другой (и более эффективный) способ записи приведенных выше примеров, в этом опять же можно убедиться, когда мы расмотрим понятие области действия.

Пространства имен и область действия

Обратимся снова к первому примеру из предыдущего раздела.

```
<!-- В этом документе не используется область действия. -->
<xdoc xmlns:sch="http://www.mudville-schools.org">
   <sch:class-list>
       <sch:class>Advanced Basket Weaving</sch:class>
       <sch:class>3D Art</sch:class>
       <sch:class>Remedial Reading</sch:class>
   </sch:clas-list>
</xdoc>
```

Здесь только одно пространство имен. Стоит ли повторять его префикс в каждом последующем элементе?

Используя область действия, можно переписать уже рассматривавшийся пример в следующем виде:

```
<xdoc xmlns="http://www.mudville-schools.org">
   <class-list>
       <class>Advanced Basket Weaving</class>
       <class>3D Art</class>
       <class>Remedial Reading</class>
   </class-list>
</xdoc>
```

Таким образом мы зафиксировали пространство имен по умолчанию как `http://www.mudville-schools.org`. Правила области действия устанавливают, что все объекты, которые встречаются в содержании элемента, находятся в его пространстве имен, *если только* атрибут или дочерний элемент *не* имеют префикса. Как видно, в соответствии с этим правилом элемент `xdoc` и все его дочерние элементы удовлетворяют этому условию, так как относятся к пространству имен `http://www.mudville-schools.org`.

Но как поступить с теми элементами и атрибутами, которые *не* принадлежат пространству имен `http://www.mudville-schools.org`?

Давайте теперь обратимся ко второму примеру и перепишем его в следующем виде:

```
<xdoc
   xmlns="http://www.mudville-schools.org"
   xmlns:chem="http://www.chem-lang.org">
   <class-list>
       <class>Advanced Basket Weaving</class>
       <class>3D Art</class>
       <class>Remedial Reading</class>
       <class>
           OrganicChemistryI <chem:class>Esters</chem:class>
       </class>

       <class>
           OrganicChemistryII <chem:class>Benzenes</chem:class>
       </class>
```

```
    </class-list>
</xdoc>
```

Здесь вся запись находится внутри пространства имен `http://www.mudville-schools.org`, исключая элементы с префиксом `chem:`; они принадлежат пространству имен `http://www.chem-lang.org`. Обратите внимание на различия в описании двух атрибутов пространств имен:

```
<xdoc
    xmlns="http://www.mudville-schools.org"
    xmlns:chem="http://www.chem-lang.org">
```

Элемент `xdoc` находится в пространстве имен `http://www.mudville-schools.org`, но не в пространстве имен `http://www.chem-lang.org`, поскольку в описании

```
xmlns="http://www.mudville-schools.org"
```

префикс в явном виде отсутствует, а это означает, что данное описание объявляет пространство имен по умолчанию, имеющее для элемента `xdoc` и его дочерних элементов приоритет над описанием `xmlns:chem`.

Атрибуты и пространства имен

Считается, что атрибуты, если у них нет другого префикса, находятся внутри пространства имен своих элементов. То есть в немного модифицированной версии первого примера предыдущего раздела:

```
<!-- Внимание: в данном синтаксисе предпочтительно использование области действия.
-->
<xdoc xmlns:sch="http://www.mudville-schools.org">
    <sch:class-list>
        <sch:class sch:type="simple">Advanced Basket Weaving</sch:class>
        <sch:class type="simplest">3D Art</sch:class>
        <sch:class>Remedial Reading</sch:class>
    </sch:class-list>
</xdoc>
```

И первый, и второй атрибуты `type` записаны верно. Они содержатся внутри одного и того же пространства имен, а именно `http://www.mudville-schools.org`.

Совет *Могут возникнуть проблемы с использованием глобальных атрибутов в XML и SGML. Второй атрибут type может быть как глобальным атрибутом, так и атрибутом элемента class, и нет способа различить эти два случая.*

Мы могли бы написать следующее выражение:

```
<xdoc
    xmlns="http://www.mudville-schools.org"
    xmlns:chem="http://www.chem-lang.org">
    <class-list>
        <class type="simple">Advanced Basket Weaving</class>
        <class chem:type="simple">Esters II</class>
```

```
        <class>Remedial Reading</class>
    </class-list>
</xdoc>
```

где элемент `class` поселка Мадвилл наследует атрибут `chem:type`, относящийся к области химических языков. Поскольку пространство имен, представленное префиксом `chem:`, предотвращает путаницу двух разных атрибутов `type`, допустима даже такая запись:

```
<class type="simple" chem:type="simple>Esters II</class>
```

Наследование не меняет того факта, что элемент `class`, о котором идет речь, представляет собой элемент `class` поселка Мадвилл.

Вывод элементов из области действия

Обращение к спецификации пространств имен бывает необходимо только в том случае, когда от агента пользователя требуется, чтобы тот проделал определенные действия с выбранными элементами. (См. ниже раздел «Зачем нужны пространства имен».) По-видимому, в приведенных выше примерах перед агентом ставилась задача проверить, удовлетворяют ли наши элементы некоторой схеме и не требуется ли от них какой-либо ответной реакции. Учтите, что для него эта работа – тяжкий труд. Предположим, что мы решили использовать некоторое пространство имен, например пространство имен HTML, а в середине его вставить некий несложный правильный участок кода, написанный на языке XML, который не должен соответствовать никакому пространству имен.

Пространства имен и область действия позволяют нам провести эту вставку. Если в описании пространства имен по умолчанию идентификатор URI пуст, то есть если:

```
<mydoc xmlns="">
    <class> Some Class </class>
</mydoc>
```

то элементы без префикса находятся вне всякого пространства имен. Другими словами, ни элемент `mydoc`, ни элемент `class` не принадлежат никакому пространству имен.

Следующий пример взят из спецификации:

```
<?xml version="1.0"?>
<Beers>
<!-- Пространством имен по умолчанию является пространство имен HTML. -->
    <table xmlns="http://www.w3.org/TR/REC-html140">
        <tr>
            <td>Name</td>
            <td>Origin</td>
            <td>Description</td>
        </tr>
        <tr>
<!-- Избавляемся от пространства имен HTML в ячейках таблицы. -->
            <td>
```

```
                    <brandName xmlns="">Huntsman</brandName>
                </td>
                <td>
                    <origin xmlns="">Bath, UK</origin>
                </td>
                <td>
                    <details xmlns="">
                        <class>Bitter</class>
                        <hop>Fuggles</hop>
                        <pro>Wonderful hop, light alcohol, good summer beer</pro>
                        <con>Fragile; excessive variance pub to pub</con>
                    </details>
                </td>
        </tr>
    </table>
</Beers>
```

Внутри этого документа использовано пространство имен HTML. Если агент знает, что делать с элементами таблицы HTML, он может применить к ним стили из таблицы стилей или преобразовать их в электронную таблицу. Однако внутри таблицы мы все равно вернемся к правильному XML-документу, который выделен в примере курсивом.

Зачем нужны пространства имен

Во «Введении» мы рассказали о некоторых вариантах применения пространств имен. Продолжим это обсуждение.

Уникальное определение элементов и атрибутов

В этом как раз и состоит основное назначение пространства имен. Все, на что в действительности претендует данная спецификация, – предоставить соответствующий способ, способный проделать данную операцию. В этой же спецификации, используя URI, предлагается определять элементы и атрибуты уникальным способом. Как приложение будет обрабатывать уникальный URI – дело программного обеспечения. Наша задача – рассмотреть два возможных способа использования пространства имен.

Повторное использование схем

Слово «схемы» во множественном числе здесь использовано преднамеренно, несмотря на то, что единственной «официальной» схемой в настоящее время является определение типа документа.

Как мы уже успели убедиться, хорошую схему написать трудно, поэтому было бы просто замечательно иметь возможность использовать уже внедренные в практику разработки.

Если имеется элемент или атрибут, принадлежащий пространству имен определенной схемы, то анализатор с проверкой на состоятельность сможет установить, соответствует ли данный элемент указанной схеме.

При этом, однако, следует учесть, что идентификатор URI в описании пространства имен *не обязательно* является тем самым URI, куда агент пользователя должен отправиться и найти DTD, в соответствии с которым следует проверить элементы на состоятельность. Должен существовать другой, пока еще не разработанный, механизм, который позволит агенту выполнить эту операцию. Возможно, программные продукты через какое-то время будут иметь встроенные средства для достижения этой цели.

В самом деле, скорей всего Web-браузер будущего будет иметь несколько встроенных схем, позволяющих проверить DTD, например, по HTML, математике, химии, музыке, для следующего, основанного на XML, поколения HTML. Безусловно, браузеры и другое программное обеспечение пользователя смогут проверить документ и на предмет соответствия схемам, определенным пользователем. (См. Web-страницу http://www.w3.org/Markup/Activity.html.)

Обучение агента пользователя

Во «Введении» было отмечено, что разметка может иметь реальный смысл только для вполне определенного агента пользователя или для человека, который понимает, к чему она относится. Продолжая уже знакомый пример с гемоглобином, `<hemoglobin>14.2</hemoglobin>`, мы можем поставить ему в соответствие пространство имен, которое предоставляет средства интерпретации этой информации, например, `www.hypermedic.com/results/hemoglobin`. Где-нибудь на этом URI могли бы находиться сведения, приведенные в табл. 4.1.

Таблица 4.1. Гипотетическая таблица реакций агента

Диапазон содержания гемоглобина	Что означает	Ожидаемое поведение агента пользователя
>17	Возможно обезвоживание. Серьезная опасность сердечного приступа или смерти	0
16.1–17	Угрожающе высокий	1
13.1–16	Нормальный диапазон	3
11.1–13	Анемия тип=слабая	3
09.1–11	Анемия тип=умеренная	3
07–09	Анемия тип=сильная	2
<7	Возможна угроза для жизни	1
<4	ТЯЖЕЛОЕ СОСТОЯНИЕ	0

Понятно, что строки с текстовым содержанием предназначены для человека. Агент пользователя будет обращать внимание только на целые числа в последнем столбце, которые повлекут за собой ответные действия с его стороны.

Лаборатория может хранить все результаты по исследованию гемоглобина в XML-документе, составленном следующим образом:

```
<meddoc xmlns= "www.hypermedic.com/results/hemoglobin">
   <hemoglobin date= "Sept2" id= "h001">14.2</hemoglobin>
```

```
    <hemoglobin date= "Sept2" id= "h002">12.2</hemoglobin>
    <hemoglobin date= "Sept2" id= "h003">16.2</hemoglobin>
...
</meddoc>
```

Теперь соответствующий агент сможет прочитать весь документ, найти указанный URI (напоминаем, что механизм для этого еще предстоит создать) и использовать находящуюся там информацию для интерпретации результатов по исследованию крови.

Чтобы получить всю необходимую информацию, этот гипотетический агент соотнесет значения `id` с историями болезни пациентов. Затем он обратится непосредственно к лечащим врачам и доложит обо всех опасных отклонениях от нормы (поведение агента пользователя = 1), а также распечатает их (поведение агента пользователя = 2) для проверки лаборантом. Все результаты, свидетельствующие о непосредственной угрозе жизни, будут немедленно доложены в центр скорой помощи, где уже будет приниматься окончательное решение.

Таким образом, пространства имен позволили «научить» агента, другими словами, направить его туда, где он «узнал» смысл результатов.

Немного воображения, и за рамками этого примера вы ясно различите широкую перспективу других возможных применений подобной методики.

Чего не может пространство имен

Хотя подобный механизм в своем описании использует URI, это еще не значит, что информация о том, как должны быть интерпретированы элементы документа, может быть получена на указанном сайте. Очевидно, что для соответствующего применения пространства имен агент должен знать, как его использовать, либо суметь отыскать информацию на этот счет.

Цель настоящей спецификации состоит в том, чтобы идентифицировать определенные элементы и атрибуты как принадлежащие некоторому пространству имен, и не более того. Вопросы, как сопоставить пространство имен и указанный сайт, как интерпретировать информацию на этом сайте и что делать после обработки информации, – предлагается решать конкретным агентам. Ниже мы поговорим о некоторых способах, с помощью которых эти операции могут быть выполнены, но подчеркиваем – пока это будут всего лишь вероятные решения. Надеемся, что будущие стандарты определят, наконец, конкретные, и официально согласованные механизмы.

Ожидаемое поведение агента пользователя

Новая спецификация пространств имен является лишь первым шагом в многоступенчатом процессе. В этом разделе будет рассмотрен вопрос, каким должно быть поведение агента, чтобы оно соответствовало текущей спецификации. Далее будет затронута проблема, связанная с ожидаемым поведением агента в будущем.

В настоящее время агент обязан только ставить в соответствие каждому элементу или атрибуту то или иное пространство имен, чтобы сделать их уникальными. В будущем от агента, возможно, будут ожидать приобретения специальных «знаний» о своем вероятном поведении. Эта перспектива может реализовываться через встроенный механизм, аналогичный подобной операции в HTML в имеющихся браузерах, или обеспечиваться установленным «протоколом обучения».

В будущем агент, вероятно, будет иметь возможность обращаться к упомянутому авторитетному источнику, чтобы узнать, как ему вести себя в той или иной ситуации. В приведенном выше примере с гемоглобином агент, скорее всего, обратится к набору «ответных реакций», которые он должен реализовать, если встретит элемент из пространства имен `http://www.hypermedic.com/results/hemoglobin`; когда же ему попадется элемент, принадлежащий пространству имен `http://www.hypermedic.com/results/electrolyte`, ему придется обратиться к другому набору. Третий перечень команд понадобится в том случае, если агент наткнется на элемент из пространства имен `http://www.harvardlabs.org/hemoglobin`.

Еще одним примером может быть обращение браузера, поддерживающего расширяемый язык таблиц стилей (XSL), к сайту для определения того, каким образом тэги могут быть воспроизведены в качестве потоковых объектов.

Применение пространств имен

Пространства имен уже используются достаточно широким кругом XML-приложений. Мы рассмотрим только три из них.

Таблицы стилей в Internet Explorer 5

В версии Beta1 браузера IE5 пространства имен используются для применения стилей к тэгам XML в HTML-документе. Подробнее об этом будет сказано в главах 7 и 8. В текущем разделе мы ограничимся только короткими примерами.

Во время разработки версии Beta1 браузера Microsoft IE5, проект спецификации пространств имен консорциума W3C, к сожалению, часто менялся. В версии Beta1 разработчики компании Microsoft, как указано ниже, использовали свой собственный вариант описания таких пространств с применением пустого тэга. Возможно, что когда вы будете читать эту книгу, пространства имен будут описываться в виде, совместимом с данной спецификацией.

В следующем примере показано, как применить каскадные таблицы стилей (CSS) к тэгам XML, находящимся внутри HTML-документа. Еще один пример вы найдете в главе 8. Пока не обращайте внимания на детали кода; все, что нас в данный момент интересует – это те элементы, которые используют синтаксис пространств имен.

```
<HTML>

<xml:namespace prefix="myns"/>

<STYLE TYPE="text/css">
```

```
    .wroxstyle1{
        display:block;
        padding-top:36pt;
        font-size:56pt;
        font-weight:bold;
        text-align:center;
        background-color:red;
        color:yellow;
        }

    .wroxstyle2{
        display:block;
        padding-bottom:36pt;
        font-size:30pt;
        font-weight:bold;
        text-align:center;
        background-color:red;
        color:yellow;
        }
</STYLE>

<BODY BGCOLOR="white" id="body">
   <P>An example of an using a namespace to apply style to an XML tag</P>

   <xtag1>The default display property for an unknown tag is inline.</xtag1>
   <xtag2>As is demonstrated here</xtag2>

   <myns:xtag class="wroxstyle1">Style sheets</myns:xtag>
   <myns:xtag class="wroxstyle2">for HTML and XML</myns:xtag>

</BODY>
</HTML>
```

Обратите внимание на то, каким образом приложение узнает, что элемент `myns:xtag` принадлежит пространству имен `xml:namespace`, в результате чего он применяет к его содержанию стиль CSS. Это пример «образованного» представителя пользователя.

Если вы просмотрите эту запись с помощью браузера IE5, то плучите на экране изображение, аналогичное рис 4.1.

Расширяемый язык таблиц стилей

Новый синтаксис расширяемого языка таблиц стилей (XSL) описан в главе 8. Язык XSL использует пространства имен, чтобы идентифицировать их и как свои собственные, и как форматирующие словарные тэги.

Вновь вернемся к нашему первому примеру, написанному на XSL. Этот код заимствован из главы 8. Повторяем, не обращайте особого внимания на детали кода; просто посмотрите, как используются пространства имен.

```
<stylesheet
```

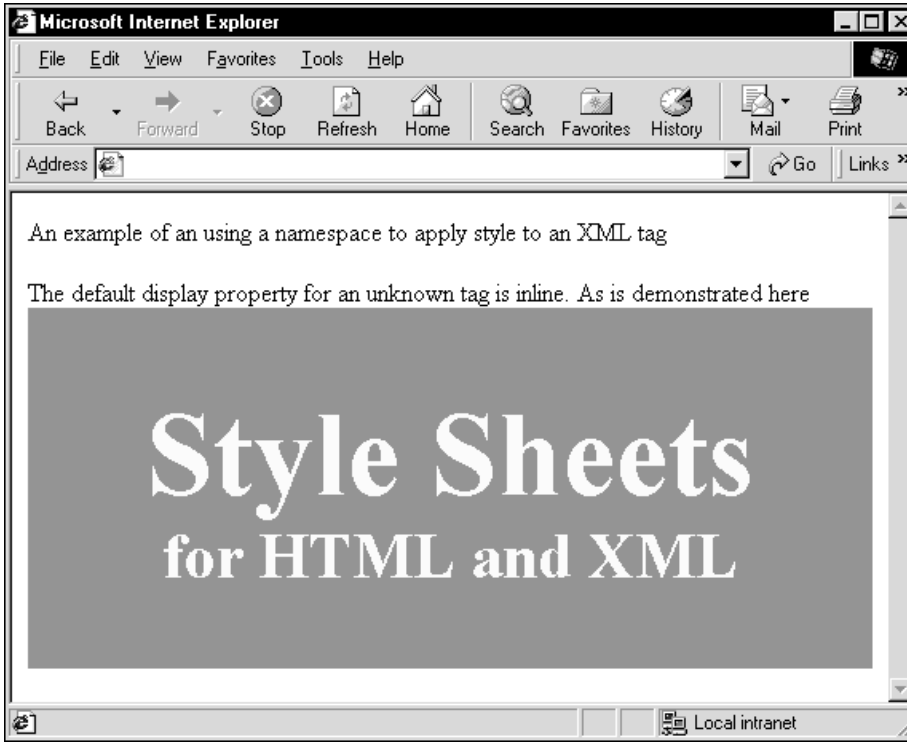


Рис. 4.1. Пример на каскадные таблицы стилей в Internet Explorer 5.0

```
xmlns:xsl="http//www.w3.org/TR/WD-xsl"
xmlns:fo="http//www.w3.org/TR/WD-xsl/FO"
result-ns="fo">

<xsl:template match="/">
    <fo:page-sequence
        font-family="times new roman,serif"
        font-size="12pt">
        <xsl:apply-templates/>
    </fo:page-sequence>
</xsl:template>
<xsl:template match="*">
    <fo:block
        font-family="times new roman,serif"
        font-size="12pt">
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>

<xsl:template match="sheepobject">
    <fo:block
        font-size="16pt">
```

```
            <xsl:text>YES I CAN!</xsl:text>
            <xsl:apply-templates/>
            <xsl:apply-templates/>
            <xsl:apply-templates/>
            <xsl:apply-templates/>
        </fo:block>
    </xsl:template>
</stylesheet>
```

Характерно, что все тэги с префиксом `xsl:` находятся в пространстве имен `http://www.w3.org/TR/WD-xsl`, а все тэги с префиксом `fo:` размещены в пространстве имен `http://www.w3.org/TR/WD-xsl/FO`.

Теперь «разумный» и «образованный» агент, а именно, XSL/XML-браузер-стайлер, будет знать, какой конкретно набор правил и действий к какому элементу применять.

Конечно, мы могли бы использовать возможности, предоставляемые областью действия, чтобы записать документ в более элегантной форме. Ниже показано, как будет выглядеть этот пример, если применить `http://www.w3.org/TR/WD-xsl` в качестве пространства имен по умолчанию. Воспроизводиться документ будет точно так же, как и в первом случае.

```
<stylesheet
    xmlns ="http//www.w3.org/TR/WD-xsl"
    xmlns:fo="http//www.w3.org/TR/WD-xsl/FO"
    result-ns="fo">

    <template match="/">
        <fo:page-sequence
            font-family="times new roman,serif"
            font-size="12pt">
            <xsl:apply-templates/>
        </fo:page-sequence>
    </template>

    <template match="*">
        <fo:block
            font-family="times new roman,serif"
            font-size="12pt">
            <apply-templates/>
        </fo:block>
    </template>

    <template match="sheepobject">
        <fo:block
            font-size="16pt">
            <text>YES I CAN!</text>
            <apply-templates/>
            <apply-templates/>
            <apply-templates/>
            <apply-templates/>
        </fo:block>
```

```
    </template>
<stylesheet>
```

Использование пространств имен в таблицах стилей позволяет агенту пользователя определять, какие элементы несут информацию и о каких именно типах команд.

Теперь давайте рассмотрим, какое значение для метаданных имеют пространства имен.

Формат описания ресурсов RDF

Формат описания ресурсов (RDF) представляет собой стандарт, в котором особенно интенсивно используется методика пространства имен. Несмотря на то, что вся информация, находящаяся в Web, доступна для чтения машинами, они, в общем-то, не понимают, что читают. RDF является только первым шагом в направлении, на котором машины, в конце концов, научатся «понимать» содержание Web. «Разбираться» в содержании необходимо для того, чтобы обеспечить совместимость между приложениями, которые обмениваются понимаемой машинами информацией.

Новейшая спецификация RDF находится на сайте http://www.w3.org/TR/WD-rdf-syntax.

Основы формата описания ресурсов

Здесь мы не станем подробно описывать, что представляет собой RDF. Наша цель – дать общее представление о том, как он используется и с какой целью.

RDF как формат описания ресурсов – это в первую очередь метаданные, то есть структурированные данные о данных, описывающие свойства элементов, их значения и взаимоотношения друг с другом. RDF предназначен для повышения эффективности поиска информации и доступа к ней.

Рассмотрим следующее предложение, фиксирующее взаимоотношения между издательством Wrox и его Web-страницей:

«Издательство Wrox является создателем Web-страницы http://www.wrox.com»

В терминологии RDF любой объект Web, такой, например, как идентификатор URI, в приведенном выше предложении является ресурсом, а ресурсы могут описываться набором тэгов, включенным в пространство имен RDF. Ниже показано, как может быть выражен наш ресурс:

```
<RDF xmlns="http://www.w3.org/TR/WD-rdf-syntax">
   <description about="http://wrox.com"/>
</RDF>
```

Эта запись информирует, что элемент `description`, как, впрочем, и его атрибут `about`, принадлежат пространству имен `http://www.w3.org/TR/WD-rdf-syntax`, поэтому «разумный» агент, изучивший RDF, знает, как интерпретировать результаты. Чтобы иметь смысл для пользователей данных RDF, описание должно использовать разметку из пространства имен, соответствующего описываемому ресурсу.

Теперь нам может понадобиться включить информацию об издательстве Wrox

в запись таким образом, чтобы агент, занимающийся поиском информации в Web, имел возможность найти данные об издательстве Wrox.

Более того, нам может понадобиться описать издательство Wrox и некоторые из его книг агенту, который знаком с набором элементов из стандартного множества метаданных Dublin Core, или «научен» составлять каталог книг для ассоциации издательств, представляющей стандартные элементы для Web-сайтов этих организаций.

Совет *Новую информацию о стандартах метаданных Dublin Core вы можете узнать на http://purl.org/metadata/dublin_core.*

Пространством имен для ассоциации издательств может быть, например, `http://www.bigpublishers.org`. Издательство Wrox вправе использовать элементы этого пространства, чтобы рассказать о своем собственном Web-сайте.

Приведенный ниже XML-документ иллюстрирует сказанное. Может случиться так, что две или более организации, строящие метаданные для описания содержимого своих архивов, используют похожие имена для тэгов с различным предназначением. Пространства имен предотвратят путаницу в интерпретации в том случае, когда файлы с метаданными из многих источников будут сведены вместе (например, поисковой программой, осуществляющей поиск во многих доменах). Следующий RDF-файл с метаданными содержит информацию об издательстве Wrox, подходящую как для каталога, совместимого со стандартом Dublin Core, так и для ассоциации издательств.

```
<RDF xmlns= "http://www.w3.org/TR/WD-rdf-syntax"
   xmlns:dc=http://purl.org/metadata/dublin_core#
   xmlns:pub="http://www.publishers.org">
   <!-- Данный тэг находится в пространстве имен rdf. -->
   <description about="http://wrox.com">
       <!-- Эти тэги находятся в пространстве имен publishers. -->
       <pub:webpage>
           <pub:url>www.wrox.com</pub:url>
           <pub:sitetitle>Wrox Press</pub:sitetitle>
           <pub:sitecontent type="computer">Computer Books
           </pub:sitecontent>
       </pub:webpage>
       <pub:booktitles>
       <pub:book authors="multiple" isbn="1525">
           <pub:title>Professional XML Application</pub:title>
           <pub:description>How to make XML work for you</pub:description>
           <dc:Creator>
               <rdf:Seq ID="CreatorAlphabeticalBySurname">
               <rdf:li> Boumphrey, Frank </rdf:li>
               <rdf:li> Hollander, Dave </rdf:li>
               <rdf:li> others</rdf:li>
               </rdf:Seq>
           </dc:Creator>
       </pub:book>
```

```
        <pub:book authors="Frank Boumphrey" isbn="1657">
            <pub:title>Professional Style Sheets for HTML and XML
            </pub:title>
            <pub:description>How to make your Web pages look great.
            </pub:description>
        </pub:book>
        </pub:booktitles>
    </description>
</RDF>
```

Без пространств имен самые полезные в практическом смысле описания метаданных оказались бы невозможны. Эти пространства позволили нам создать единый документ, который содержит описание информации, понятной как человеку, так и агенту пользователя, знающему RDF, а также программам, совместимым со стандартом Dublin Core, или тэгами ассоциации издательств.

Заключение

В этой главе была рассмотрена одна из самых потенциально важных спецификаций в группе XML. Надеемся, читатели успели убедиться, каким образом пространства имен дают программистам возможность повторно использовать уже примененные схемы и повышать уровень своего рода «образованности» Web-приложений.

Мы рассмотрели синтаксис представленной в этой главе спецификации, разобрались с тем, как использование области действия придает особую элегантность XML-документам. Мы узнали, каким образом можно переключаться с одного пространства имен на другое, и даже как попасть в «нулевое» пространство имен.

Несмотря на то, что данная спецификация, казалось бы, нацелена лишь на уникальное определения элементов и атрибутов в Web, ее применение открывает перед пользователями широкие, а главное, многообещающие в практическом смысле перспективы.

Наконец, мы рассмотрели три главных направления, в которых уже сейчас используются пространства имен.

Глава 5. Ссылки и указатели в XML

В этой главе мы рассмотрим, каким образом формируются ссылки в XML, и чем они отличаются от ссылок в HTML. Стандарт создания ссылок был сконструирован отдельно от основной спецификации XML 1.0, тем не менее, перечень требований в этой части является важной составной частью всего комплекта стандартов для XML. В настоящее время существуют две разработки, определяющие формат ссылок в XML. Обе находятся на стадии рабочего проекта и именуются:

- спецификация *X-ссылок* (Xlink), устанавливающая стандарт, в соответствии с которым отдельные документы должны ссылаться друг на друга. Вы можете найти ее по адресу http://www.w3.org/TR/WD-xlink;
- язык *X-указателей* (Xpointer), предназначенный для совместного использования со спецификацией XLink в целях адресации структур внутри XML-документов. Его описание можно найти по адресу http://www.w3.org/TR/WD-xptr.

В этой главе будут также рассмотрены:

- простые ссылки (simple links);
- расширенные ссылки (extended links);
- использование указателей для выделения частей документа.

Спецификация *X-ссылок* (Xlink) и язык *X-указателей* (Xpointer) ранее были объединены в одном стандарте, который назывался «Язык создания ссылок в XML» (XML Linking Language, XLL), и этот трехбуквенный акроним (Three Letter Acronym, TLA) в качестве обобщающего термина до сих пор используется в литературе. Вы встретитесь с ним и в этой главе.

Кроме того, сообщение о принципах построения спецификации *X-ссылок*, датированное мартом 1998 года, находится на сайте

http://www.w3.org/TR/NOTE-xlink-principles

Есть веские основания полагать, что окончательные рекомендации консорциума W3C по этому вопросу вряд ли будут значительно отличаться от мартовского рабочего проекта, несмотря на то, что спецификации подобного рода по мере их проработки обычно претерпевают серьезные изменения. Недавно была сформирована рабочая группа по ссылкам в XML, которая вплотную занялась оформлением стандартов, регулирующих использование тех или иных приемов и методик в этой области.

Совет *Другими словами, стандарт консорциума W3C еще не внедрен, поэтому большая часть материала этой главы относится к разряду «Вот будет замечательно, когда они это реализуют!». Однако прежде чем вы махнете рукой и решите пропустить эту главу, вспомните нехитрую житейскую мудрость, которая гласит, что лучше раньше, чем позже; тем более, изложенный здесь материал поможет вам легко разобраться с различными способами создания ссылок в XML. Пусть даже сегодня подобная информация может и не иметь практического значения, знания в этой области способны серьезно повлиять на ваше отношение к созданию документов, когда дело дойдет до внедрения XML-приложений.*

Теперь остановимся вкратце на процессе формирования ссылок в HTML. Детальное рассмотрение этого вопроса впоследствии поможет читателям разобраться в серьезных различиях между XML и HTML.

Формирование ссылок в HTML

В настоящее время можно с уверенностью утверждать, что первоначальный успех формата HTTP по сравнению, например, с форматом FTP, в основном объясняется легкостью, с которой в нем можно сослаться на другой документ.

Эта операция в HTML выполняется с помощью тэга `<A>`. Тип ссылки, используемой в HTML, в XML называется простой ссылкой.

Простые ссылки

Здесь представлены два HTML-документа минимально возможного размера, которые должны находиться в одной и той же папке или директории:

ch05_docA.htm

```
<HTML>
<A HREF= "ch05_docB.htm"> Эта гиперссылка доставит вас в документ ch05_docB.htm</A>
</HTML>
```

ch05_docB.htm

```
<HTML>
<A HREF= "ch05_docA.htm"> Эта гиперссылка доставит вас в документ ch05_docA.htm</
A>
</HTML>
```

Загрузите оба документа в одну и ту же директорию, откройте один из них, и щелкните по гиперссылке. Теперь можно часами переходить из одного документа в другой и обратно.

Немного терминологии

В следующем абзаце вы познакомитесь с терминологией, связанной с созданием ссылок. Позже мы подробно рассмотрим все понятия, а также приведем их формальные определения. Для начала этого будет вполне достаточно.

Ссылки, о которых мы только что говорили, называются *простыми ссылками* (simple links). *Ссылка* (link) – это то, по чему можно щелкнуть мышкой. При щелчке агент пользователя ищет *локатор* (locator) или *адрес* (address), который представляет собой URL (например, `ch05_docA.htm`). Щелчок по ссылке переносит вас из одного *ресурса* (resource) в другой. Ресурс – это документ или часть документа. Заметим, что оба файла `ch05_docA` и `ch05_docB` являются, как их обычно называют, *участвующими ресурсами* (participating resources). При щелчке по ссылке в документе `ch05_docA` документ `ch05_docA` действует как *источник* (source), а документ `ch05_docB` – как ее *место назначения* (destination). Такая ссылка является *встроенной* (inline), поскольку она служит в качестве одного из своих собственных ресурсов. (Позже в этой главе, при изучении расширенных ссылок XML, мы увидим, что такое *внешняя* (out-of-line) ссылка.)

Заметим, что существует различие между связыванием по ссылке и адресацией, которое мы рассмотрим в следующем разделе.

Различие между связыванием и адресацией

Связывание (linking) и *адресация* (addressing) относятся к терминам, которые путают чаще других и, случается, используют один вместо другого, хотя между ними, при всей их схожести, есть довольно тонкое отличие.

Когда кто-то утверждает, что два предмета связаны ссылкой, все, что ему остается – это установить взаимоотношение между ними. Например, издательство Wrox и Фрэнк Бумфрей связаны тем, что Фрэнк Бумфрей пишет книги для издательства Wrox. Только в этом смысле можно говорить о связывании.

Адресация, в свою очередь, относится к поиску объекта, на который ссылаются. Как Фрэнк Бумфрей, так и издательство Wrox имеют электронную почту и почтовый адрес, у них есть Web-сайты со своими собственными доменами и URL. Если у нас есть эта информация, мы знаем, *как* установить связь между двумя объектами.

В простом примере, приведенном выше, документы `ch05_docA` и `ch05_docB` связаны между собой. Кроме того, они имеют собственные адреса – идентификаторы URI. Поскольку документ `ch05_docA` содержит адрес документа `ch05_docB` и наоборот, ссылка является простой. Если бы мы для получения этой информации должны были обратиться к другому документу или ресурсу, то ссылка была бы расширенной. В HTML не существует прямого способа создания расширенных ссылок.

Указатели в HTML

Как уже было сказано, указатель используется для адресации *внутри* документа. Используя те или иные средства, он определяет специальную область внутри документа. В следующем примере, который можно просмотреть с помощью браузера IE4, использован идентификатор `id` конкретного элемента.

Короткий HTML-документ, приведенный ниже, называется `ch05_docD.htm`. Обратите внимание, как идентифицируются абзацы с помощью `id`.

```
<HTML>
<A HREF= "ch05_docC.htm">Эта гиперссылка приведет вас в ch05_docC.htm</A><BR>
"Lorem ipsum и т.д."
```

```
<P ID="a1">Это первый абзац с id = a1</P>
"Lorem ipsum dolor и т.д."
<P ID="a2"> Это первый абзац с id = a2</P>
"Lorem ipsum dolor sit и т.д."
<P ID="a3"> Это третий абзац с id = a3</P>
"Lorem ipsum dolor sit amet и т.д."
</HTML>
```

Совет *Пожалуйста, не пытайтесь понять, что значит "Lorem ipsum dolor sit amet...". Это вспомогательный или подставной текст, используемый в издательском деле для заполнения частей документа на начальном этапе верстки, до получения фактического материала.*

А вот еще один документ со ссылками и указателями, которые обеспечивают переход к частям документа, используя `id` в качестве идентификаторов. Документ называется `ch05_docC.htm`.

```
<HTML>
<BR><A HREF= "ch05_docD.htm#a1">Эта гиперссылка приведет вас к первому абзацу
документа ch05_docD.htm</A>
<BR><A HREF= "ch05_docD.htm#a2"> Эта гиперссылка приведет вас ко второму абзацу
документа ch05_docD.htm </A>
<BR><A HREF= "ch05_docD.htm#a3"> Эта гиперссылка приведет вас к третьему абзацу
документа ch05_docD.htm </A>
</HTML>
```

Теперь щелчок по любой из ссылок приведет вас в соответствующий абзац нужного документа.

Совет *На самом деле, этот документ слишком мал, чтобы можно было заметить эффект от подобного способа, поскольку весь он помещается на одной странице. Чтобы воочию увидеть действие указателя, добавьте в абзацы какой-нибудь текст.*

Часть URL, начинающаяся со знака `#`, представляет собой идентификатор отрывка, которым в данном случае является атрибут `id` конкретного элемента.

В синтаксисе XML ссылка будет выглядеть следующим образом: `<A HREF= "ch05_docD.htm#id(a1)">` (обратите внимание на появление `id`, сообщающего нам тип указателя). Использование `id` как указателя возможно только в браузерах IE4.x.

Заметим, что при использовании синтаксиса XML не требуется никаких особых средств, чтобы попасть в определенное место документа `ch05_docD`: строка `"ch05_docD.htm#id(a1)"` содержит всю необходимую информацию для нахождения заданного места. Вместо документа `ch05_docD` можно указать любой документ, в котором есть элементы с атрибутами `id`. При этом нет необходимости иметь физический доступ к файлу `ch05_docD` и вставлять в него специальный объект, на который можно сослаться.

Посмотрите, насколько этот прием отличается от знакомого вам синтаксиса:

```
<A NAME="a1"></A>
```

В подобном случае необходимо иметь физический доступ к файлу `ch05_docD`, чтобы поместить туда метку для создания ссылки. Эта метка является не указателем, а некоторым уникальным и специальным объектом, который нужно поместить в документ, чтобы впоследствии попасть в обозначенное место.

В синтаксисе XML при указании на `id` вы также называете место в документе с определенным `id`, то есть применяете свойство, которое используется и для других целей. Если документ `ch05_docD` имеет тэги с `id`, то для указания на определенное место внутри него не требуется доступ к этому документу.

Сказанное можно объяснить на простом примере. Уникальным в указателе является то, на что он указывает. Прогуливаясь с ребенком, я могу привлечь его внимание к листку на земле и попросить подобрать. Когда же мне приходится использовать метод типа `NAME`, я вынужден объяснять: «Там-то и там-то упал листик, я подойду, помечу его специальным тэгом, а потом ты поищи и принеси его мне».

Вскоре будет описано, каким образом указатели в XML позволяют указать на что-либо внутри документа, не имея к нему доступа.

Теперь, когда мы кратко просмотрели знакомый материал и в какой-то степени освоились с терминами, связанными с языком XLL, перейдем к ссылкам и указателям в XML. Учтите, что по мере продвижения вперед нам придется все чаще и подробней знакомиться с терминологией. Если вы предпочитаете иметь под рукой полноценные объяснения тех или иных понятий, можете прямо сейчас изучить раздел, посвященный обзору терминологии.

Простые ссылки в XML

Простые ссылки в XML очень похожи на ссылки в HTML. Однако в то время как знакомый с HTML агент пользователя знает, что тэг `<A>` представляет собой тэг ссылки («якорь» (anchor) в терминологии HTML), в случае XML необходимо сообщить агенту пользователя, что данный тэг является тэгом ссылки. Необходимо также сообщить ему, каким именно типом ссылки он является.

Определение тэгов ссылки

Рассмотрим следующий правильный XML-документ.

```
<linkdoc>
   <para>Comment can be found at <mylink href="DocumentA.htm>Document
   </mylink>where all
               is explained
       </para>
   <para>
           <mylink href="DocumentB.htm">Document B</mylink> can be found here
       </para>
</linkdoc>
```

Очевидно, что он содержит ссылки как на документ `DocumentA`, так и на доку-

мент DocumentB. Очевидно и то, что <mylink> является тэгом ссылки, а значение атрибута href представляет собой адрес или, точнее, URL ресурса, на который необходимо сослаться.

Однако агент пользователя XML различить эти нюансы не способен. С его точки зрения мы с таким же успехом могли бы назвать тэг <wroxlink>, <address> или даже <timbuctoo>. Поэтому нам нужен способ, с помощью которого можно проинформировать агента пользователя о том, что этот фрагмент должен быть интерпретирован как ссылка.

В браузере, совместимом со спецификацией XLink, мы сможем это сделать, используя зарезервированный атрибут XML xml:link. Он может принимать следующие значения: simple (простая), extended (расширенная), group (групповая), locator (локатор), document (документ).

Совет *Значение simple описывает ссылку, похожую на ту, которую в HTML можно создать с помощью тэга <A>. Остальные значения представляют собой концепции, новые для HTML-сообщества, и будут довольно подробно описаны ниже.*

Добавляя зарезервированный в XML атрибут xml:link="simple", получим следующее:

```
<linkdoc>
   <para>Comment can be found at
          <mylink xml:link="simple" href="DocumentA.htm">Document A</mylink>
where all is explained
       </para>
   <para>
          <mylink xml:link="simple" href="DocumentB.htm">Document B</mylink> can
be found here
       </para>
</linkdoc>
```

Теперь браузер, знакомый со спецификацией Xlink, сможет понять, что при щелчке мышью по тексту подразумевается переход по соответствующему адресу.

Чтобы обеспечить состоятельность документа, необходимо описать атрибуты ссылки в определении типа документа. Это можно сделать так:

```
<!ELEMENT mylink (#PCDATA)>
<!ATTLIST mylink
   xml:link        CDATA              #FIXED          "simple"
   href            CDATA              #REQUIRED
>
```

Преимущество такого подхода, конечно же, заключается в том, что, описывая атрибут xml:link как #FIXED, мы не обязаны включать его в тэг. Его присутствие всегда подразумевается.

Проницательный читатель может задать вопрос: «А как насчет атрибута href? Я не вижу при нем никаких префиксов XML.»

На самом деле в спецификации XLink можно найти несколько предопределенных имен для атрибутов. Эти атрибуты в значительной степени расширяют роль даже простой ссылки по сравнению с возможностями тэга <A> в HTML. В этом мы убедимся в следующем разделе.

Атрибуты, предлагаемые спецификацией XLink

Ниже представлен список предлагаемых атрибутов с их разрешенными значениями. Вместе взятые, они могут расширить возможности применения даже простой ссылки. Помните, что язык XML различает заглавные и строчные буквы.

```
xml:link   (simple|extended|group|locator|document)   'simple'
href       CDATA                      #REQUIRED
inline (true|false)                   'true'
role       CDATA                      #IMPLIED
title      CDATA                      #IMPLIED
show       (embed|replace|new)        #IMPLIED
actuate    (auto|user)                #IMPLIED
behavior   CDATA                      #IMPLIED
```

Рассмотрим каждый из них по очереди.

Атрибут xml:link

Как мы уже говорили, простая ссылка в XLink ведет себя также, как и в HTML. Другие значения атрибута `xml:link` мы подробно рассмотрим в этой же главе.

Атрибут href

Этот атрибут указывает на локатор, который в основном состоит из адреса. В него входят идентификатор URI и X-указатель или просто URI. X-указатели будут детально рассмотрены позже во второй части этой главы. В табл. 5.1 приведены несколько примеров допустимых значений атрибута `href`, с которыми вы должны уже были встречаться.

Таблица 5.1. Допустимые значения атрибута href

Ресурс в той же директории	documentB.htm
Ресурс в той же директории и X-указатель	documentB.htm#a1
Удаленный ресурс	http://www.hypermedic.com/documentC.htm
Удаленный ресурс и X-указатель	http://www.hypermedic.com/documentC.htm#a1

Атрибут inline

Этот атрибут для всех простых ссылок должен иметь значение `true` (истинно). Большинство других ссылок также являются встроенными (inline). Позже мы рассмотрим внешние (out-of-line) ссылки, для которых значение атрибута `inline` равно `false` (ложно).

Атрибут role

Это сервисный атрибут, который сообщает агенту пользователя значение ссылки. Значение атрибута `role` (роль), представляющее собой строку, в большинстве случаев применяется агентом пользователя как средство предоставления дальнейшей информации о месте назначения, на которое указывает эта ссылки, до щелчка по ней.

Атрибут title

Название этого атрибута говорит само за себя (`title` = заголовок). Он отображает заголовок документа, на который установлена ссылка. Одно из предлагаемых применений атрибута `title` состоит в том, чтобы агент пользователя показывал его значение при наведении курсора мыши на ссылку.

Атрибут show

Если этот атрибут принимает значение `replace` (заместить), то старый ресурс замещается новым. Это традиционный метод в HTML.

Если атрибут `show` (показать) принимает значение `new` (новый), то открывается новое окно. Это можно сделать и в HTML с помощью простого скрипта.

Если атрибут `show` принимает значение `embed` (внедрить), то новый ресурс будет внедрен в старый. Это можно сделать в HTML с помощью *очень длинного* скрипта и браузера IE4.x.

В следующем примере

```
<linkdoc>
   <para>Comment can be found at
       <mylink xml:link="simple" href="DocumentA.htm" show="embed">document
A</mylink>
       where all is explained
   </para>
</linkdoc>
```

документ `DocumentA` будет внедрен в документ-источник предположительно в том месте, где нужно щелкнуть по ссылке в документе-источнике.

Атрибут actuate

Атрибут `actuate` (приводить в действие) используется с расширенными ссылками. По мере поиска одного из ресурсов расширенной ссылки, ресурс, присоединенный к ссылке со значением атрибута, равным `auto` (автоматически), также будет найден. Если значение атрибута равно `user`, то для того, чтобы отыскать ресурс, требуется некоторое действие со стороны пользователя. Процесс станет более понятным, когда мы перейдем к разбору расширенных ссылок.

Атрибут behavior

Атрибут `behavior` (поведение) описывает, что может произойти после щелчка по ссылке (в процессе перехода к другому ресурсу). Пользователю может быть предоставлено новое окно, выпадающий список и так далее. Возможные типы поведения определяются агентом пользователя, однако `behavior` может указывать на предлагаемые значения. Атрибут `behavior` предоставляет место для описания требуемого поведения.

Совместимый со спецификацией XLink агент пользователя предположительно будет иметь встроенный распознаватель этих атрибутов, но, тем не менее, чтобы документ считался состоятельным, они должны быть описаны в DTD.

Абсолютно необходимыми атрибутами ссылки являются только `xml:link` и `href`.

Атрибут xml:attribute

Возможно, нам понадобится использовать в качестве элемента ссылки элемент, который уже имеет атрибут с одним из только что перечисленных имен. Наиболее очевидный кандидат в такой ситуации – атрибут `title`. В этом случае спецификация XLink дает нам возможность переименовать «известный» атрибут с помощью атрибута `xml:attribute`. Соответствующий процесс называется *переименованием* (remapping) атрибута.

Следующий пример взят из мартовской спецификации, на которую мы уже ссылались ранее.

Пусть имеется элемент `<text-book>` (учебник), которому я хочу присвоить атрибуты `title` и `role`. Значением `title` будет заголовок учебника, а значением `role` – его назначение. Другими словами, то ли учебник (text-book) будет использован как основной для курса, который я преподаю, то ли как вспомогательный.

Описания элемента и атрибута могут выглядеть следующим орбразом:

```
<!ELEMENT text-book ANY>
<!ATTLIST text-book
   title   CDATA                    #IMPLIED
   role    (PRIMARY|SUPPORTING)     #IMPLIED
>
```

Теперь я хочу сделать этот элемент элементом ссылки, но атрибуты `title` и `role` уже описаны. Что же делать?

На выручку приходит атрибут `xml:attribute`. Если взять имя атрибута спецификации XLink по умолчанию (`role`, `href`, `inline`, `title`, `show`, `actuate`, `behavior`, `steps`) и имя, на которое оно заменяется (отделенное пробелом) в качестве значения, то далее на него можно ссылаться из спецификации XLink по новому имени. Если требуется более одного замещения, значение атрибута `xml:attribute` может состоять из нескольких таких пар, разделенных пробелом.

Теперь второе имя может быть использовано вместо имени, взятого из XLink по умолчанию. Когда агент пользователя встретит второе имя, он автоматически преобразует его в первое.

Список атрибутов теперь будет выглядеть следующим образом:

```
<!ATTLIST text-book

   title       CDATA                        #IMPLIED
   role        (PRIMARY|SUPPORTING)   #IMPLIED
   xml:link        CDATA                              #FIXED "simple"
   xml:attributes  CDATA
                   #FIXED "title xl-title role xl-role"
>
```

Когда поддерживающий спецификацию XLink агент пользователя встретит следующий код:

```
<text-book title="Professional Style Sheets"
    role="PRIMARY"
    xl-title="Primary text book for the course"
    xl-role="ONLINE-PURCHASE"
    href="HTTP://www.wrox.com"/>
```

он распознает атрибуты `xl-title` и `xl-role` как замены атрибутов `role` и `title`, описанных в спецификации XLink.

В заключение нашего краткого обзора следует сказать, что в своей самой безыскусной реализации простая ссылка XML очень похожа на элемент ссылки `A` в HTML. Элемент ссылки определяется агентом пользователя по зарезервированному атрибуту `xml:link`. Если эта ссылка имеет только атрибут `href`, она в точности совпадает со ссылкой `A` в HTML. Другие атрибуты спецификации XLink расширяют функциональность ссылки. Особенно это относится к атрибуту `show`, который уточняет, каким образом должен быть показан ресурс, к которому ведет ссылка.

Совместимые с XLink агенты пользователя

Что касается уже изученного материала, а также того, который еще только предстоит рассмотреть, проблема у них одна и та же: не существует агента пользователя, который мог бы реализовать данную спецификацию. К сожалению, это беда всех новых спецификаций. Однако следует учесть, что использование в документе XLink не мешает применять для создания ссылок и другие перечни требований, а любая их реализация остается делом агента пользователя.

Поскольку еще не существует даже браузера, совместимого с XML, что говорить о браузере, совместимом со спецификациями XLink или XPointer!

Обзор терминологии

Прежде чем обсуждать расширенные ссылки, обратимся к определению тех или иных понятий, применяемых в этой области, но теперь в более строгом – формальном – виде.

Ниже приводятся выдержки из раздела 1.3 «Терминология» мартовской, 1998 года, спецификации. К некоторым из них даны комментарии и пояснения авторов сборника.

Цитата

Дерево элементов (element tree).
Представление соответствующей структуры, определенной тэгами и атрибутами в XML-документе, основанное на понятии «рощи» в соответствии с определением стандарта ISO DSSSL.

Точное объяснение значения этого термина будет дано в главе 6, посвященной объектной модели документа (DOM) XML. Здесь же этот материал будет рассмот-

рен не так подробно, а только в той части, которая касается указателей.

Цитата *Встроенная ссылка (inline link).*
Вообще говоря, ссылка, которая служит одним из своих собственных ресурсов. Точнее, ссылка, содержание элемента которой является участвующим ресурсом. Ссылка A в HTML, clink в HyTime и xref в TEI являются примерами встроенных ссылок.

Если ссылка действительно встроена в документ, который можно прочитать, и если в нем можно щелкнуть мышью по чему-нибудь и куда-нибудь попасть, то в таком случае обычно говорят о встроенной ссылке. Если же ссылки и адреса содержатся в отдельном ресурсе, то речь ведут о внешней ссылке. Надеемся, что примеры, приведенные в разделе, посвященном расширенным ссылкам, прояснят ситуацию.

Цитата *Ссылка (link).*
Явное взаимоотношение, установленное между двумя или более объектами данных или их частями.

Заметим, что ссылку следует отличать от адреса. Ссылка просто объявляет о наличии взаимоотношений между двумя ресурсами.

Цитата *Ссылочный элемент (linking element).*
Элемент, который утверждает наличие этих взаимоотношений и описывает их характеристики.

Цитата *Локальный ресурс (local resource).*
Содержание встроенного ссылочного элемента. Заметим, что содержание ссылочного элемента может быть в явном виде указано с помощью обычного локатора в том же элементе ссылки. В этом случае ресурс считается не локальным, а удаленным.
В примере

<P> Нажмите <A HREF="somedoc.htm"> здесь</A>

слово «здесь» является локальным ресурсом.

Цитата *Локатор (locator).*
Данные, которые предоставляются как часть ссылки и указывают на ресурс.

Локатором обычно является URL или идентификатор фрагмента.

Цитата *Многонаправленная ссылка (multidirectional link).*
Ссылка, прохождение которой может быть инициировано более, чем одним из участвующих в ней ресурсов. Заметим, что возможность возврата после прохождения однонаправленной ссылки не делает эту ссылку многонаправленной.

Цитата *Внешняя ссылка (out-of-line link).*

Ссылка, содержание которой не служит в качестве одного из ее участвующих ресурсов. Такие ссылки предполагают необходимость дополнительного средства, например, группы расширенных ссылок, которое указывает приложению, где искать ссылки. Внешние ссылки обычно требуются для поддержки многонаправленных обходов и для предоставления возможности ресурсам, доступным только для чтения, иметь исходящие ссылки.

Цитата *Участвующий ресурс (participating resource).*

Ресурс, который принадлежит ссылке. Потенциально в ссылке участвуют все ресурсы; участвующий ресурс делает это явным образом для конкретной ссылки.

Цитата *Удаленный ресурс (remote resource).*

Любой участвующий в ссылке ресурс, который указывается с помощью локатора.

Цитата *Ресурс (resource).*

В самом общем смысле это адресуемый сервис или единица информации, которая участвует в ссылке. Например, файлы, изображения, документы, программы, результаты запросов. Конкретно, ресурс – это все, что можно получить с помощью локатора, в каком-либо ссылочном элементе. Заметим, что этот термин и его определение взяты из основных спецификаций, управляющих World Wide Web.

Если у некоторого объекта есть адрес, которым можно воспользоваться, чтобы найти этот объект, щелкнув мышью по «горячей точке» (hotspot), то такой объект является ресурсом. Он может быть изображением, базой данных, другим документом или даже кодом, который допускается использовать для создания какого-либо изображения. «Горячая точка» (hotspot) – это старый термин мультимедиа. В документе так называют любое место, которое реагирует определенным образом, когда его выделяют с помощью щелчка кнопкой мыши или указания курсором.

Цитата *Подресурс (sub-resource).*

Часть ресурса, на которую указывается как на точное место назначения ссылки. Например, ссылка может требовать, чтобы весь документ был найден и продемонстрирован, но определенная часть (части) этого документа может представлять собой данные, на которые ссылаются особо, и они должны быть обработаны характерным для приложения образом: выделением, прокруткой и так далее.

На самом деле востребованной обычно бывает только часть документа. Эта часть и будет называться подресурсом. См. первый пример в разделе, посвященном X-указателям.

Цитата *Прохождение (traversal).*

Процесс использования ссылки, то есть доступа к ресурсу. Прохождение может быть инициировано действием пользователя (например, щелчок мышью по демонстрируемому браузером содержанию элемента ссылки) или произойти под управлением программы.

Другими словами, прохождение – это переход от одного ресурса к другому.

Расширенные ссылки

Простые ссылки действительно не отличаются особой сложностью. Находясь в одном из ресурсов, мы щелкаем по ссылке и, как правило, либо оказываемся в другом ресурсе, либо получаем его. Простые ссылки исключительно универсальны, и, как мы уже говорили ранее, возможно, именно они являются причиной всеобщего признания Web и ее быстрого взлета.

Однако простые ссылки тоже не свободны от определенных ограничений. Рассмотрим следующий отрывок из HTML-документа:

```
<P>This is the report of a debate on the origins of the cold war.</P>
<P>
   <BR><A HREF="intro.htm">Introduction</A>
   <BR><A HREF="propos.htm">Proposal</A>
   <BR><A HREF="rebut.htm">Rebuttal</A>
   <BR><A HREF="comment.htm">Commentary</A>
   <BR><A HREF="summary.htm">Summary</A>
</P>
```

То же самое место в XML-документе могло бы выглядеть следующим образом:

```
<para>This is the report of a debate on the origins of the cold war.</para>
<para>
   <debate-link xml:link="simple" href="intro.xml">Introduction</debate-link>
   <debate-link xml:link="simple" href="propos.xml">Proposal</debate-link>
   <debate-link xml:link="simple" href="rebut.xml">Rebuttal</debate-link>
   <debate-link xml:link="simple" href="comment.xml">Commentary</debate-link>
   <debate-link xml:link="simple" href="summary.xml">Summary</debate-link>
</para>
```

На рис. 5.1 показано, в каком виде брузер мог бы продемонстрировать любой из приведенных выше фрагментов документа.

Конечно, сразу видно, что этот метод достаточно удобен и в достаточной мере удовлетворяет нашим целям, однако спецификация XLink предоставляет еще более эффективный способ обработки подобных материалов с помощью расширенных ссылок.

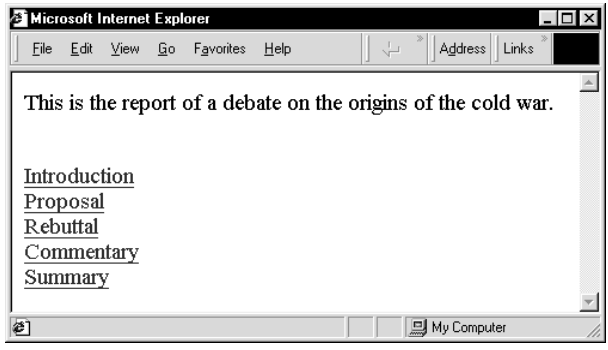


Рис. 5.1. Демонстрация в браузере текста с простыми ссылками

Встроенные расширенные ссылки

Вот как приведенный выше пример выглядел бы с использованием встроенной расширенной ссылки.

```
<xdoc>
<para>
This is the report of

   <mylink             xml:link="extended"              inline="true"
title="Debate">
       <locator  xml:link="locator"   href="intro.xml"   title="Introduction"/>
       <locator  xml:link="locator"   href="propos.xml"  title="Proposal"/>
       <locator  xml:link="locator"   href="rebut.xml"   title="Rebuttal"/>
       <locator  xml:link="locator"   href="comment.xml" title=:Commentary"/>
       <locator  xml:link="locator"   href="summary.xml" title="Summary"/>

   a debate

   </mylink>

on the origins of the cold war.

</para>
</xdoc>
```

Обращаем ваше внимание на следующие моменты:

- теперь у нас есть единственный ссылочный элемент `mylink`, атрибуты которого принимают следующие значения: `xml:link='extended'`, `inlinelink='true'` и `title='Debate'` (дискуссия);
- сам по себе этот элемент довольно бесполезен, он похож на пустое здание офиса – выглядит хорошо, но без сотрудников и оборудования пользы от него никакой. Все его обязанности заключаются в размещении рабочих элементов. В этом примере подобным объектом является элемент `locator`, который дает адреса и информацию для ссылок;

- элементы `locator` в данном примере имеют атрибуты `xml:link="locator"`, `href=[URI]`, `title=[cdata]`;
- элемент `locator` можно было бы назвать и по-другому, также как и элемент `debate-link`, который мы использовали в примере с простой ссылкой. Мы назвали его `locator` просто для того, что бы подчеркнуть его предназначение;
- заметим, что это встроенная ссылка, поскольку она действует как один из своих собственных ресурсов, а именно как текст `a debate` (дискуссия), по которому можно щелкнуть;
- ссылочный элемент расширенной ссылки также может содержать другие элементы, а именно элементы `group` и `document`, которые мы рассмотрим позже;
- ссылочный элемент `locator` может иметь другие атрибуты;
- ссылочный элемент расширенной ссылки может иметь и другие атрибуты; их тоже рассмотрим позже, как и атрибуты предыдущего элемента.

Способ, с помощью которого знакомый с XML браузер будет обрабатывать ссылочный элемент расширенной ссылки, целиком от него же и зависит. Возможно, при загрузке страницы он просто покажет ссылки, что приведет к результату, очень похожему на тот, с которым мы встречались в приведенном выше простом примере. Возможно, браузер продемонстрирует ссылки в тот момент, когда пользователь щелкнет по содержанию ссылочного элемента расширенной ссылки, а текст документа будет воспроизведен заново с обтеканием ссылок.

Скорее всего, программное обеспечение создаст выпадающее окно, которое позволит пользователю сделать дальнейший выбор, как показано на рис. 5.2. (Обратите внимание на то, что это изображние сконструировано; XML-браузер с такими возможностями пока еще не существует.)

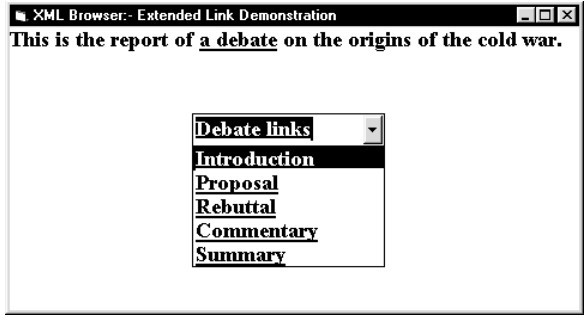


Рис. 5.2. Текст с расширенной встроенной ссылкой

Как видите, в этом примере заголовок окна ссылок показывает значение атрибута `title` элемента `mylink`, а выпадающий список – значения атрибутов `title` соответствующих элементов `locator`.

Если бы мы добавили атрибут `role`, то при указании курсором мыши на ссылку мог бы появляться текст атрибута `role`, например, в небольшом желтом окне, но такие подробности, конечно же, остаются на усмотрение агента пользователя. Спецификация просто допускает возможность существования подобных уточнений.

Внешние расширенные ссылки

В предыдущем примере ссылка являлась встроенной, поскольку существует нечто, по чему можно щелкнуть, а именно строка `a debate` (дискуссия). Однако, как будет показано ниже, вполне возможна такая ситуация, когда ссылки находятся в другом документе.

Заметим, что приведенный ниже XML-файл является состоятельным, хотя в нем нет ничего такого, что можно было бы продемонстрировать. Все что делает этот файл – определяет ссылки на различные документы и предоставляет их адреса.

```
<xdoc>

<!-- Ниже приведен пример внешней расширенной ссылки. -->

   <mylink                          xml:link="extended"        inline="false"
title="Out-of-line example">
       <locator xml:link="locator"   href="DocA.htm"  title="Document A"/>
       <locator xml:link="locator"   href="DocA.htm"  title="Document B"/>
       <locator xml:link="locator"   href="DocA.htm"  title="Document C"/>

   </mylink>

<!-- Заметим, что эта ссылка НЕ является ресурсом. Данный файл содержит ссылки на
различные документы. -->

</xdoc>
```

Заметим также, что указанные в примере документы могут и не подозревать, что на них созданы ссылки.

Похожая ситуация встречается и в обыденной жизни. Моя дочь постоянно твердит о Леонардо ди Каприо, а я ей отвечаю: «Да, помню, в молодости Шон Коннери у меня с языка не сходил».

Эти фразы каким-то образом связывают Лео и Шона, но я уверен, что никто из них не подозревает о нашем к ним интересе. Однако вряд ли кто-нибудь осмелится опровергнуть мое утверждение, что им обоим прекрасно известно, как, вообще говоря, публика ими интересуется, и в принципе кто-то может сравнивать их друг с другом.

У меня есть Web-сайт www.hypermedic.com – я никогда не упускаю возможности отрекламировать его и при этом уверен, что ссылающиеся на него сайты безусловно существуют, но пока мне случайно не удастся наткнуться на такую сылку, я об этом не узнаю.

Эти примеры иллюстрирует схема, приведенная на рис. 5.3.

Заметим, что весь поток информации направлен в одну сторону. Обратного потока, вытекающего из документа, на который ссылаются, нет; более того, ни один из документов не подозревает, что на него ссылаются.

«Хорошо, – скажете вы, – но какая же от этого польза? Как это способствует выполнению наших задач?»

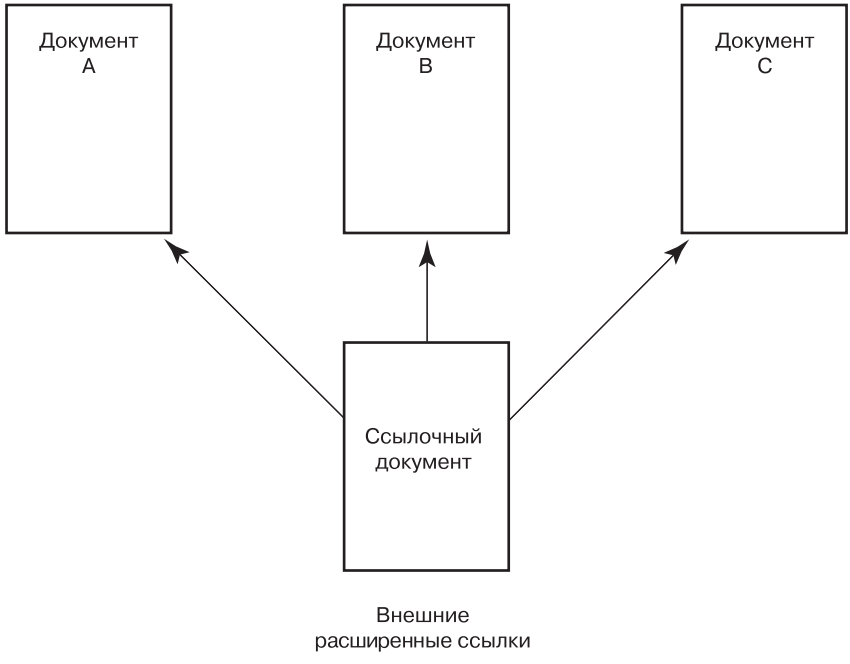


Рис. 5.3. Схема связывания документов внешними ссылками

Ниже приведено, что говорится по данному поводу в спецификации.

Цитата *Ключевой вопрос, связанный с использованием внешних ссылок, заключается в том, каким образом соответствующее приложение могло бы находить эти ссылки и управлять ими, особенно когда они хранятся отдельно от тех документов, где находятся их участвующие ресурсы.*

Факт состоит в том, что сама по себе внешняя ссылка достаточно бесполезна. Но в сочетании с подходящим агентом пользователя она предоставляет огромные возможности. Подобная перспектива как раз и станет предметом нашего дальнейшего обсуждения.

Как используются расширенные внешние ссылки?

Не стоило бы в который раз упоминать о том, что в настоящее время не существует агента пользователя, способного реализовать те удивительные возможности, о которых мы собираемся поведать читателям, но, может быть, вы относитесь к тем из них, кто читает эту книгу именно для того, чтобы создать такое программное обеспечение.

Имея в виду подобную возможность, попробуем рассмотреть разнообразные ситуации, в которых программное обеспечение может использовать внешние ссылки.

Использование внешних расширенных ссылок

Малая сеть intranet

В этом случае мы имеем несколько документов, `doc 1`, `doc 2`, `doc 3`... `doc n`, ни в одном из которых нет ни одной ссылки, однако все они связаны внешними ссылками, содержащимися в нескольких документах: `link 1`, `link 2`, `link 3`...`link n`.

Предположим, мы хотим получить доступ к любым ссылкам на документ по щелчку на нем правой кнопки мыши. (Извините, мы работаем на PC. Можете обвинять нас в платформизме. Ладно, вы, пользователи Macintosh с однокнопочной мышью, щелкните кнопкой мыши, нажав одновременно клавишу **Shift**.)

Программное обеспечение должно поддерживать базу данных со всеми ссылками. Предположим, пользователь щелкнул правой кнопкой мыши на документе `doc m`. Оказалось, что существует два документа с внешними ссылками.

Первая из них находится в группе документов, называемой `Annual Report` (годовой отчет), которая содержится в файле `report-links.xml`:

```
<xdoc>

   <mylink xml:link="extended" inline="false" title="annual report">
       <locator xml:link="locator" href="docM.xml" title="Document M"/>
       <locator xml:link="locator" href="sales.htm" title="Quarterly sales
       _reports"/>
       <locator xml:link="locator" href="production.xml" title="Quarterly
       _Production output"/>
   </mylink>

</xdoc>
```

Вторая ссылка содержится в группе документов, называемой `Think Tank` (мозговой центр) и хранящейся в файле `think-tank-links.xml`:

```
<xdoc>

   <mylink xml:link="extended" inline="false" title="Think Tank">
       <locator xml:link="locator" href="docM.xml" title="Document M"/>
       <locator xml:link="locator" href="predict.htm" title="Future Widget
       _needs"/>
       <locator xml:link="locator" href="market.xml" title="Marketing
       _strategies for widgets."/>
   </mylink>

</xdoc>
```

В сети intranet, которая может содержать как XML- так и HTML-документы, каждый из них уже просмотрен программным обеспечением на предмет поиска ссылок. Эта процедура повторяется всякий раз при добавлении нового документа. Каждая ссылка на документ `Document M` будет обнаружена, а список ссылок будет выдан в поле этого документа.

Если ссылка на `Document M` простая, то база данных просто напечатает файл, в котором содержится такая ссылка.

Вот, например, выдержка из HTML-документа `Document C`.

```
<P> Comments on company progress can be found in <A HREF= "docM.xml">Document</A>
where future trends are discussed. </P>
```

Информация базы данных о ссылках на `Document M` может выглядеть так, как указано в табл. 5.2.

Таблица 5.2. Ссылки на Document M

Название	URI	Тип	Связанные ссылки
Document C	docC.htm	Простая	Нет
Think Tank (мозговой центр)	think-tank-links.xml	Расширенная	Sales.htm, production.htm
Annual Report (годовой отчет)	report-links.xml	Расширенная	predict.htm, market.xml

Вполне вероятно, что после щелчка правой кнопкой мыши по документу `Document M` появится аналогичная таблица, где пользователь сможет увидеть сами ссылки после щелчка по соответствующему URI.

Хотя для малой сети intranet такой подход и хорош, в большой сети количество ссылок скоро выйдет из-под контроля, и уж конечно же, никто не станет поддерживать подобную базу данных для всей Web. Это приводит нас ко второму примеру – большой сети intranet.

Большая сеть intranet

Очевидно, что для большой сети Intranet управление ссылками легче всего осуществить с помощью расширенных внешних ссылок. Понятно, что при этом варианте при щелчке правой кнопкой мыши по документу пользователь получит огромное число ссылок.

К счастью, в спецификации XLink для этого случая имеются типы ссылок `group` и `document`.

В случае большой Intranet нам по-прежнему нужна центральная база данных для всех ссылок, которая пополняется всякий раз, когда в сети появляется новый документ. Однако теперь каждый документ включает также тип ссылки `group`. Например, `Document J` может иметь следующий элемент ссылки:

```
<docgroup xml:link="group">
   <doc xml:link="document" href="links1.xml"/>
   <doc xml:link="document" href="links2.xml"/>
   <doc xml:link="document" href="links3.xml"/>
</docgroup>
```

Совет *Заметим, что это очень похоже на сочетание элемента расширенной ссылки с элементом, содержащим локатор. Здесь docgroup – элемент, в котором содержатся ссылки, а doc – элемент, содержащий URI.*

Эти элементы сообщат агенту пользователя, что интересующие его ссылки содержатся в указанных документах, которые включают в себя (предположительно) внешние ссылки.

Теперь, когда мы щелкаем правой кнопкой мыши по документу, агент пользователя, вместо того чтобы обращаться к базе данных, просто покажет ссылки из элемента `docgroup`, и затем использует указанные пути, чтобы найти соответствующие файлы.

Теперь зададимся вопросом, как сделать, чтобы агент пользователя мог проследить всю линию ссылок. Например, файл `links1.xml` может содержать ссылку на документ `report.htm`, который также имеет несколько ссылок на URI. Те, в свою очередь, содержат еще больше ссылок, в результате очень скоро у нас скопится такое количество информации, которое невозможно обработать. Весьма вероятно, что при переходе от одной ссылки к другой, затем к третьей и т. д. очень скоро программа либо будет заблокирована, либо зациклится. Для предотвращения подобной ситуации в спецификацию языка XML введен атрибут `steps`, который сообщает агенту пользователя, на сколько уровней следует продвигаться при просмотре. Поэтому, если бы в указанном выше примере мы записали:

```
<docgroup xml:link="group" steps="2">
   <doc xml:link="document" href="links1.xml"/>
   <doc xml:link="document" href="links2.xml"/>
   <doc xml:link="document" href="links3.xml"/>
</docgroup>
```

то агент пользователя знал бы, что при просмотре ссылок ему не следует продвигаться более чем на два уровня.

В данном случае мы имеем доступ ко всем документам, о которых идет речь. В конце концов, они принадлежат нашей собственной сети Intranet. А что было бы, если бы мы вообще не имели доступа к документам? Чем могли бы нам помочь расширенные ссылки в таком случае? Скоро мы вернемся к этому вопросу.

Поведение агента пользователя

Несмотря на то, что этот вопрос имеет мало отношения к рассматриваемой теме, авторы полагают уместным пояснить, как ведет себя агент пользователя в рассмотренных примерах.

Известно, что спецификация ограничивается описанием стандартного набора правил, которым должен следовать агент пользователя, а также стандартного языка, введение которого в документ вызывает определенную реакцию агента пользователя. Спецификация не дает рекомендаций по поводу того, как агент пользователя должен обрабатывать групповые ссылки (соответствующие элементу `docgroup`). Все, что в ней сообщается – это указание, что ссылки каким-то образом сгруппированы вместе.

Один агент пользователя может предоставить выпадающий список ссылок, а другой – их краткий обзор.

Повторяем, в спецификации ничего не сказано о том, как агент пользователя должен обрабатывать внешние ссылки – только то, что эти документы связаны. Различные агенты пользователя могут демонстрировать данную информацию различными

способами. Один – при щелчке правой кнопкой по документу показывать выпадающий список, другой – предоставлять строку состояния с такой же информацией.

Дистанционное комментирование документов

Предположим, что конкурирующая компания Rival поместила в Internet некоторый документ, и мы хотим, чтобы руководители нашей компании Acme смогли прочесть его и прокомментировать в такой форме, чтобы их комментарии были доступны другим сотрудникам нашей компании. Как этого добиться?

Вот документ, о котором идет речь:

```
<rivaldoc>

   <para>Our company is the greatest.</para>
   <para>Acme company is the worst.</para>
   <para>We float like a butterfly and sting like a bee.</para>
   <para>Acme is ugly.</para>
   <para>We are going to bury Acme.</para>
   <para>Our widgets are the greatest!</para>
</rivaldoc>
```

Конечно же, у нас нет доступа к исходным файлам документов компании Rival, хотя мы и можем просматривать тексты этих докумнтов. На рис. 5.4 показано, как выглядит приведенный выше документ в нашем браузере.

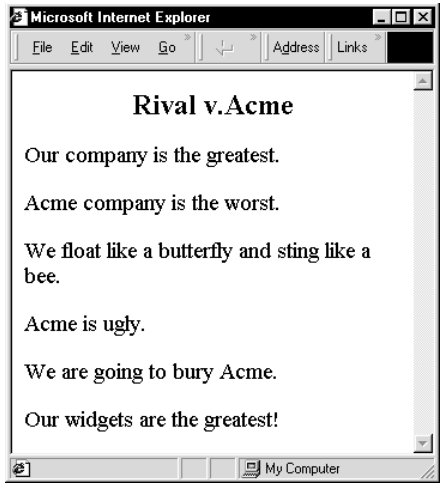


Рис. 5.4. Просмотр документа в браузере

У нас есть специальное программное обеспечение, которое позволяет вставлять комментарии в приведенный выше документ. При загрузке документа компании Rival, наш агент пользователя подставляет версию, дающую возможность нашим руководителям вносить различные комментарии. Для каждого руководителя можно зафиксировать отдельный символ. Все, что он должен сделать – это щелкнуть по документу и записать свои замечания, практически так же, как это делается в редакторе Word. Затем символ становится частью документа в нашем источнике

данных. Другие руководители смогут прочесть комментарии, щелкнув мышью на этих символах. Конечно, если компания Rival действительно умна, она будет регулярно обновлять свой документ, обесценивая таким образом наши усиия.

После того, как руководители Acme поработают на документом, он станет похож на рис. 5.5.

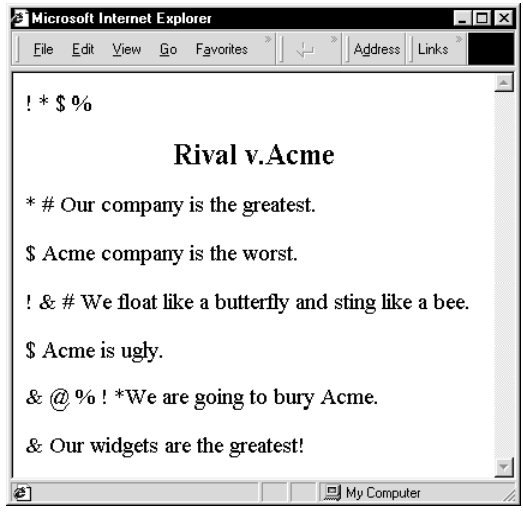


Рис. 5.5. Просмотр откомментированного документа

Каждый символ определяет ссылку на комментарий одного из руководителей. Знак доллара представляет собой ссылку на мнение Боба Чистоганнера, знак **#** – на отношение Джо Льстивера, и так далее. Щелкая по соответствующему символу, каждый из сотрудников имеет возможность познакомиться с любыми комментариями.

Как агент пользователя хранит и организует комментарии – это дело агента пользователя. Однако один из способов, которым он может воспользоваться, состоит в том, чтобы хранить каждый комментарий в отдельном документе. Доступным он становится, когда пользователь щелкает по соответствующему символу. Этот комментарий может или встраиваться в первоначальный документ, что приведет к обновлению вида всего документа, или появляться в отдельном окне.

Обслуживание ссылок

Сильнейшую головную боль для Web-мастера доставляют оборванные ссылки. Если на нашем Web-сайте тысяча страниц, триста из которых имеют ссылку на Ассоциацию производителей безделушек, то когда эта Ассоциация изменит свой URL, Web-мастеру придется вручную переписать каждую из этих трехсот страниц.

Совет *Конечно же, существует программное обеспечение, которое автоматизирует эту задачу.*

Можно поместить все ссылки в один документ и ссылаться на них из своих страниц. Тогда пользователю придется менять только один документ. Это очень похоже на концепцию таблиц стилей, определений типа документа или других схем и скриптов, где несколько странц ссылаются на один источник. Такой подход иллюстрирует схема, представленная на рис. 5.6.

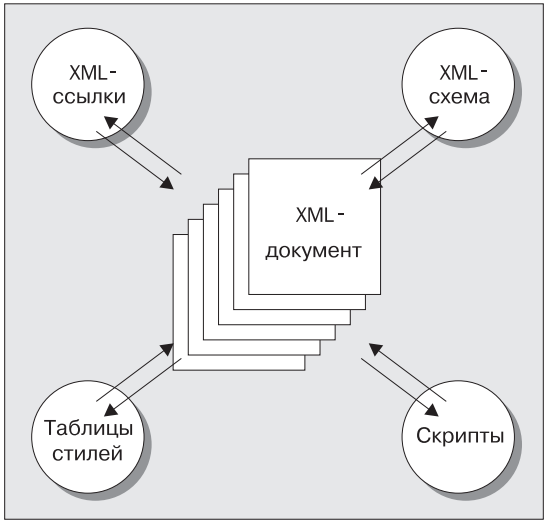


Рис. 5.6. Различные концепции ссылок

Здесь представлены всего лишь несколько примеров того, как агент пользователя может реализовать методы спецификации X-связей. Я уверен, что вы можете придумать и другие примеры.

Давайте теперь обратимся к особым свойствам, которые предлагает программистам спецификация X-указателей, а именно: к гибкости и универсальности.

X-указатели

Как было сказано выше, спецификация XLink служит для создания ссылок на различные документы, в то время как спецификация XPointer предоставляет нам способ указать на что-либо внутри документа. Мы уже видели, как браузер IE4.x использует атрибут `id` для указания на информацию, содержащуюся в документе. Этот атрибут может быть использован и в XML-документе. Его применение спецификацией поощряется.

Далее приведен пример, который, несмотря на свою упрощенность, демонстрирует возможности указателей, определенных в спецификациях XLL, а также иллюстрирует сходство и различие ссылок в HTML и XML.

```
<xDocA>
   <para id="p1">This is the first paragraph.</para>
   <para id="p2">This is the second paragraph.</para>
```

```
    <para id="p3">This is the third paragraph.</para>
    <para id="p4">This is the fourth paragraph.</para>
</xDocA>
```

В этом случае есть смысл использовать документ `DocB`, чтобы вложить в него третий абзац документа `DocA`.

Вот как выглядит код документа `DocB`:

```
<xDocB>
<para>Click on the hash marks to embed the third paragraph of Document A
_in this document

    <mylink
        xml:link="simple"
        show="embed"
        href="DocA.htm|id(p3)">
    [####]
    </mylink>
</para>

</xDocB>
```

Заметим, что в XML необходимо сообщить агенту пользователя, какого типа указатель используется в том или ином случае. В приведенном примере в качестве идентификатора используется атрибут `id` элемента. Таким образом, мы должны поместить выражение `|id(p3)` после URL. В случае, если мы пропускаем `id` и оставляем только выражение `|p3`, спецификация требует, чтобы агент пользователя подразумевал присутствие `id`, поэтому синтаксис, который используется в версии HTML браузера IE4.x, допускается. (См. следующий раздел, описывающий синтаксис локатора.)

Обратите внимание, в качестве идентификатора фрагмента вместо символа диеза (`#`) употребляется символ вертикальной черты (`|`). С его помощью можно указать агенту пользователя, что требуется восстановить не весь документ, а только ту его часть, на которую указывает идентификатор фрагмента – элемент с атрибутом `id`, равным `p3`.

Совет *Результат, конечно же, будет зависеть от того, какая реакция запрограммирована у агента пользователя на символ вертикальной черты. Той же цели можно было бы добиться, используя указатель span (интервал). Поскольку знакомство с последним вариантом значительно усложнило бы приведенный выше, достаточно простой пример, мы обсудим эту возможность позже.*

Теперь обратите внимание, как работает гипотетический агент пользователя, который демонстрирует общий вид докмента до щелчка по значкам диеза (рис. 5.7).

А на рис. 5.8 показано, как он будет выглядеть после щелчка.

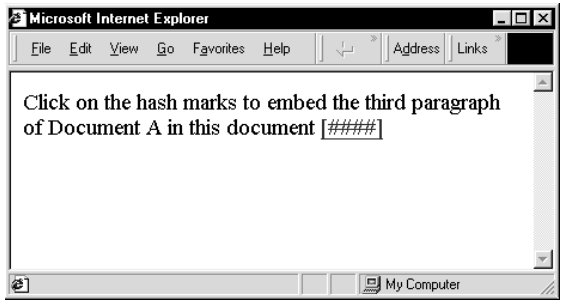


Рис. 5.7. Документ до щелчка на знаках диеза

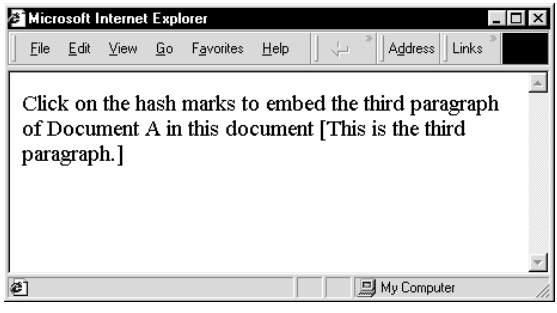


Рис. 5.8. Документ после щелчка на знаках диеза

Приведенный пример наглядно демонстрирует некоторые преимущества спецификации XLL:

- мы создали ссылку на документ;
- мы указали агенту пользователя, какое действие ему предпринять;
- мы определили некоторую область в удаленном документе, используя указатель `[|id(p3)]`;
- указателем является то, что следует за знаком диеза или вертикальной чертой;
- мы определили фрагмент как элемент с идентификатором `id`;
- мы указали на часть удаленного ресурса (подресурс), который с помощью этого же указателя хотим вложить в документ;
- получив команду, агент пользователя нашел указанный фрагмент и вложил его в текущий документ.

Это основы спецификации XPointer, но, как говорят уличные торговцы: «Подождите, у нас есть еще много-много всего...». Так что давайте рассмотрим это самое «много всего».

Синтаксис локатора

Локатор является значением атрибута `href`, то есть `href=[locator]`.

Он имеет вид URL, за которым следует соединитель – или символ вертикальной черты (`|`), или диез (`#`), а затем указатель или имя.

В спецификации XLink это выражено более формальным языком:

```
[1] Locator::=URI
| Connector (XPointer | Name)
| URI Connector (XPointer | Name)
[2] Connector ::= '#' | '|'
```

Диез (#) указывает агенту пользователя, что необходимо показать весь документ, вертикальная черта (|) сообщает, что должна быть восстановлена только часть документа (как в примере, приведенном выше). В спецификации об этом также говорится.

Цитата *Если соединителем (Connector) является вертикальная черта, это означает, что модель обработки, которая должна быть использована для доступа к назначенному ресурсу, не указана.*

URI – это, конечно же, знакомый нам URL.

Позже синтаксис XPointer будет разобран подробнее, но если, как мы уже упоминали, за соединителем следует имя, например #p3, предполагается, что это сокращение от #id(p3). Следующая цитата показывает, как это записано в спецификации.

Цитата *Если за соединителем сразу следует имя Name, то имя Name является сокращением для указателя XPointer "id(Name)"; это означает, что подресурс представляет собой элемент в содержащем его ресурсе, значение атрибута ID языка XML которого совпадает с именем Name. Сокращение используется для поощрения использования адресации с помощью id.*

Имя Name, конечно же, является любым разрешенным в XML именем (см. главу 2).

Вот некоторые примеры состоятельных локаторов.

```
DocA.htm#id(p3)
DocA.htm#p3
http://www.someserver.com/DocA.htm#p3.
```

Все приведенные выше локаторы эквивалентны, за исключением последнего, где дан полный адрес. Все они указывают, что на сервере должен быть найден весь документ A, но элемент с id="p3" каким-то образом необходимо выделить. В HTML-документе это обычно делается с помощью автоматической прокрутки элемента на начало страницы. Вот еще несколько примеров:

```
DocA.htm|id(p3)
DocA.htm|p3
http://www.someserver.com/docA.htm|p3.
```

Локатор указывает на то же самое место в документе, но сервер теперь не обязан предоставлять весь документ, возможно, он пошлет только один элемент,

как в нашем первом упрощенном примере.

Совет *Проницательный читатель мог заметить, что для того, чтобы подобная операция стала возможной, сервер должен уметь распознавать спецификацию XLL. Каким-то образом клиент должен сообщить серверу, что требуется только часть документа, и сервер, конечно же, обязан распознать запрос и, что более важно, действовать в соответствии с ним. Сомнительно, что эта часть спецификации будет внедрена в ближайшее время. Она очень перспективна, поскольку обеспечивает экономию пропускной способности каналов, благодаря пересылке только части требуемого документа. Однако, полагаю, эта часть спецификации одной из последних войдет в практику.*

Приведем еще несколько примеров:

```
DocA.htm#root().child(2,para)
DocA.htm#child(2,para)
```

Этот указатель сообщает агенту пользователя о том, что необходим второй дочерний элемент корневого элемента документа, который к тому же имеет тип `para`. Эти два локатора одинаковы.

Иными словами, локатор указывает на место, которое в следующем документе отмечено звездочкой. (Звездочка, конечно же, не является частью документа, она вставлена только для иллюстрации.)

```
<xdoc>
   <para>This is paragraph 1.</para>
   *<para>This is paragraph 2.</para>
   <para>This is paragraph 3.</para>
</xdoc>
```

Уточнение *Заметим, что в практике программирования нумерация обычно начинается с нуля (0). По некоторым причинам в спецификации XLink она начинается с 1. Поскольку в объектной модели документа (DOM) второй абзац называется para[1], это является потенциальным источником великой путаницы.*

Рассмотрев пример, иллюстрирующий работу указателей XPointer, давайте более детально рассмотрим их синтаксис.

Синтаксис X-указателей

Указатель XPointer состоит из серии *термов места* (location terms), отделенных друг от друга точкой. Каждый из них определяет место внутри документа, обычно в терминах предыдущего местоположения. Эти местоположения записываются в терминах дерева документа. Если вы не знакомы с этой концепцией, загляните в главу 6, посвященную объектной модели документа, где дано полное

описание дерева документа. При работе обычно пользуются серией термов места, которые обозначают либо *абсолютное место* (absolute location), либо *относительное* (relative), определяемое в сравнении с некоторым абсолютным местом.

Абсолютное место

Существует четыре формы указания абсолютного места в документе:

- корень (root);
- начало отсчета (origin);
- элемент с атрибутом `id`;
- местоположение типа HTML.

Ниже приведен пример на все четыре формы указания места. Давайте рассмотрим каждую из них, используя упрощенный документ `simpledoc.xml`.

```
<xdoc>
   <para>This is paragraph 1.</para>
   <para>This is paragraph 2 which contains <mylink xml:link="simple"
   _href="#p3">a link</mylink>.</para>
   <para id="p3">This is paragraph 3.</para>
   <A NAME= "bottom"/>
</xdoc>
```

Корень

Это, конечно же, корневой элемент документа (root), которым в документе из предыдущего листинга является элемент `<xdoc>`.

Внешняя ссылка, указывающая на корень данного документа, будет иметь следующий синтаксис:

```
"href=simpledoc.xml#root()"
```

В принципе, это делать не обязательно, поскольку если никакой указатель не определен, то автоматически подразумевается корень документа.

Повторим пример, приведенный выше – при этом нам придется забежать немного вперед – и запишем:

```
href= "simpledoc.xml#root().child(2,para)"
href= "simpledoc.xml#child(2,para)"
```

В обеих строках указывается на одно и то же место – на второй абзац.

Начало отсчета

Указание абсолютного места от *начала отсчета* (origin) пригодно только для указателя, отмечающего некоторую позицию в том же документе. Он ссылается на подресурс, с которого начинается прохождение. В файле `simpledoc.xml` (см. пример) элемент `<mylink>` указывает на элемент `<para>` с идентификатором `id`, равным `"p3"`.

Если значение атрибута `href` элемента `<mylink>` имело бы следующий вид:

```
href= #origin().following(1, #element)
```

то оно указывало бы на то же самое место документа – на первый элемент, следующий за началом отсчета.

Очевидно, что указание абсолютного места по типу начала отсчета может быть использовано только для указания места в том же самом документе, то есть рамки его применения достаточно узки. Однако этот способ идеален для указания на абстрактные величины, такие, например, как следующая глава того же документа.

Элемент с атрибутом id

Мы уже не раз использовали его в наших примерах. Несомненно, идентификатор `id` является наиболее устойчивым к ошибкам и требующим наименьших затрат методом указания места внутри документа. Напомним, что в нашем примере

```
<mylink xml:link="simple" href="#p3">a link</mylink>
```

значение атрибута `href` указывает на следующий параграф.

Если бы локатор находился вне документа, ссылка могла бы выглядеть следующим образом:

```
<mylink xml:link="simple" href="simpledoc.xml#p3">a link</mylink>
```

Местоположение типа HTML

Очевидно, что еще длительное время нас будет окружать множество HTML-документов, и этот абсолютный локатор создан для того, чтобы использовать их синтаксис, который имеет вид: `<A NAME="...">`.

Чтобы сослаться из какого-либо документа на элемент `<A NAME="BOTTOM">`, в нашем примере необходимо обратиться к следующему синтаксису:

```
<mylink xml:link="simple" href="simpledoc.htm#html(bottom)">a link</mylink>
```

Цитата показывает, что об этом говорится в спецификации.

Цитата

3.2.4. Ключевое слово html

Если указатель XPointer начинается с выражения html(NAMEVALUE), это означает, что истоком места (location source) является первый из элементов типа A, имеющих атрибут NAME, значение которого равно NAMEVALUE. Это в точности та же функция, которая выполняется идентификатором фрагмента "#"(диез) в HTML-документе.

В этом разделе мы рассмотрели абсолютные локаторы. Теперь перейдем к относительным. Для экономии места мы не будем рассматривать их в полном объеме, просто воспользуемся характерными примерами, чтобы каждый читатель смог познакомиться с этим методом и самостоятельно разобраться в спецификации, которая написана достаточно понятно и логично.

Относительное указание места

Указатель XPointer всегда начинается с абсолютного локатора. Если таковой не указан, то подразумевается корень документа. После абсолютного локатора сле-

дует точка, а за ней любое количество относительных локаторов, тоже отделенных друг от друга точкой.

Например, следующая запись указывает на двадцать девятый абзац четвертого подраздела третьего основного раздела истока места, которым является корневой элемент:

```
href= "somedoc.htm#root().child(3,DIV1).child(4,DIV2).child(29,P)"
```

Кстати, это то же самое, что и

```
href= "somedoc.htm# child(3,DIV1).(4,DIV2).(29,P)"
```

поскольку в тех местах, где ключевые слова опущены, предполагается, что они точно такие же, как и предшествующее ключевое слово. Но мы немного забегаем вперед. Рассмотрим, что представляют собой ключевые слова.

Ключевые слова относительного указания

Само определение этих объектов уже вносит некоторую путаницу, поскольку подобное название очень схоже (но не совпадает в точности) с так называемыми «ключевыми словами», используемыми в объектной модели документа. Мы уже обращали ваше внимание на другой потенциальный источник путаницы, связанный со спецификацией XLL, где нумерация начинается с единицы, а не с нуля.

Ниже приведены ключевые слова из спецификации, которые фактически не требуют пояснения.

Цитата

Каждое из этих ключевых слов указывает на последовательность элементов или других типов узлов XML, из которой выбирается исток места (location source). Аргументы, передаваемые ключевому слову, определяют, какие типы узлов из этой последовательности действительно выбраны. Каждое ключевое слово, которому здесь дается краткое определение, подробно описывается в следующих разделах:

- *child (дочерний). Указывает на узлы, непосредственно вложенные в исток места;*
- *descendant (потомок). Указывает на узлы, появляющиеся в любом месте внутри содержания истока места;*
- *ancestor (предок). Указывает на элементы, содержащие исток места;*
- *preceding (предшествующий). Указывает на узлы, которые предшествуют истоку места;*
- *following (следующий). Указывает на узлы, которые следуют за истоком места;*
- *psibling (предшествующий брат). Указывает на родственные узлы (имеющие того же родителя, что и исток места), которые предшествуют истоку места;*

- *fsibling (следующий брат). Указывает на родственные узлы (имеющие того же родителя, что и исток места), которые следуют за истоком места.*

Если ключевое слово опущено, считается, что оно эквивалентно непосредственно предшествующему ключевому слову; в первом терме места любого указателя XPointer (включая вложенные) ключевое слово не должно быть опущено.

Рассмотрим, как использовать эти ключевые слова для точного указания места внутри документа.

Использование ключевых слов

Общий вид обозначения указателя следующий:

#[абсолютное место].ключевое слово ([целое число], [Тип узла])

Совет *Полную информацию об узлах вы найдете в шестой главе, посвященной объектной модели документа.*

Иначе говоря, здесь используется одно из ключевых слов, а затем в качестве аргументов этого ключевого слова передаются номер его экземпляра и тип узла. Несколько примеров прояснят ситуацию. Они будут взяты из следующего XML-документа[1] (очень похожего на тот, какой используется в спецификации):

```
<?xml version= "1.0"?>
<!DOCTYPE SPEECH [
<!ELEMENT SPEECH (#PCDATA|SPEAKER|DIRECTION)*>
<!ATTLIST SPEECH
ID ID #IMPLIED>
<!ELEMENT SPEAKER (#PCDATA)>
<!element direction (#PCDATA)>
]>

<SPEECH ID="a27">
<SPEAKER>Polonius</SPEAKER>

<DIRECTION>crossing downstage</DIRECTION>

Fare you well, my lord.

<DIRECTION>To Ros.</DIRECTION>

You go to seek Lord Hamlet? There he is.

</SPEECH>
```

Но прежде рассмотрим типы узлов, распознаваемые спецификацией XLL.

[1] Полоний: *Пересекая сцену*. Желаю здравствовать, принц. *Розенкранцу*. Вам надо принца Гамлета? Он здесь.
Текст приводится по изданию: Шекспир В. Гамлет, принц Датский // Полн. собр. соч., т. 6. – М., 1960. *(Прим. ред.)*

Типы узлов

Узел может быть одного из следующих типов. Ниже приведен синтаксис из спецификации (3.3.3).

```
NodeType::= Name | '#element' | '#pi' | '#comment' | '#text' | '#cdata' | '#all'
```

Заметим, что перед всеми типами узлов, кроме типа `Name`, необходимо ставить символ `#`.

Узел Name

Узел `Name` (имя) относится к имени элемента. С помощью отрывка из Гамлета приведем пример указателя XPointer, используя узел `Name`.

```
Id(a27).child(2,DIRECTION)
```

Таким способом мы выберем второй узел с именем `DIRECTION`, то есть узел, содержащий текст `"To Ros"` (Розенкранцу). Заметим, что поскольку XML различает заглавные и строчные буквы, имя должно воспроизводиться точно.

Обратите внимание, как мы используем идентификатор `id` в качестве абсолютного локатора для нашего относительного синтаксиса.

Узел #element

Тип узла `#element` «подсчитывает» элементы XML. Приведем пример:

```
Id(a27).child(2,#element)
```

Указанное место совпадает со вторым элементом, дочерним к элементу с `id`, равным `'a27'`, то есть с первым элементом `DIRECTION`, содержание которого является текстом `"crossing downstage"` (пересекая сцену).

Обратите внимание на различие в нумерации дочерних элементов в спецификациях XLL и объектной модели документа: XLL нумерует элементы, начиная с 1, а не с 0.

Узел #pi

В примере нет команд приложений (processing instruction, PI), но если бы они были, то указатель работал бы точно так же, как и для элемента.

Узел #comment

В приведенном выше примере комментариев нет, но если бы они были, то указатель работал бы точно так же, как и для элемента.

Узел #text

Определение дочернего текста смотрите в главе 6, посвященной объектной модели документа. Коротко говоря, этот узел представляет собой любой участок, содержащий неразмеченный текст. Можете ли вы посчитать, сколько элементов дочернего текста содержится в элементе примера `SPEECH` с идентификатором `ID="a27"`?

Ответ – 4! Символы конца строки между тэгами `SPEECH` и `SPEAKER`, а также между тэгами `SPEAKER` и `DIRECTION` считаются дочерним текстом!

Поэтому запись

```
Id(a27).child(3,#text)
```

указывает на дочерний текст со значением строки, равным:

```
"
Fare you well, my lord.
"
```

Обратите внимание на включение символов конца строки в значение строки. Это XML!

Узел #cdata

В примере нет узлов типа `cdata`. Обратите внимание, как неудачно спецификация XPointer использует узел `cdata`, записывая его строчными буквами, в то время как спецификация DOM ссылается на секции CDATA. Надеемся, что в окончательном варианте этого не будет.

Узел #all

Этот тип узла «подсчитывает» все узлы, поэтому указатель

```
Id(a27).child(3,#all)
```

выберет третий дочерний узел, а именно, второй разрыв строки.

Завершая на этом наш краткий обзор указателей XPointer, рассмотрим выбор с помощью атрибута и выбор с помощью ключевых слов `string` (строка) и `span` (интервал).

Указание с помощью атрибута

Спецификация XLink позволяет указывать внутрь документа с помощью атрибута и его значения. Общий вид синтаксиса указателя для этого случая таков:

```
[ключевое слово] ([целое число], [#element | имя], [имя атрибута], [значение атрибута])
```

В нашем примере будет использовать ключевое слово `child`, имея в виду, что вместо него может поместить любое другое ключевое слово.

Заметим также, что поскольку только элемент может иметь атрибуты, единственными допустимыми типами узлов являются `#element` и `Name`.

Приведем пример:

```
child(1,#element,title, "Professional XML Applications")
```

Эта запись указывает на первый элемент, который имеет атрибут `title` со значением `"Профессиональные XML-приложения"`.

Заметим кстати, что следующие два указателя эквивалентны:

```
html(top)
```

и:

```
root().descendant(1,A,NAME, "top")
```

Шаблон

Кроме того, мы можем использовать *шаблон* (wildcard) как для атрибутов, так и для их величин. Запись:

```
child(1,#element,title, *)
```

указывает на первый элемент с атрибутом `title`, который может иметь любое значение, а запись:

```
child(1,#element, *, *)
```

указывает на первый элемент, имеющий атрибут любого вида.

Интервальный терм места

Поскольку указатель XPointer позволяет возвращать не весь документ, а только часть его (как и в том случае, когда мы используем атрибут `show='embed'`), необходимо средство, с помощью которого можно отделить данную часть от остального документа. Для этого предусмотрено ключевое слово «интервал» (`span`).

Общий вид синтаксиса следующий:

```
span ([XPointer], [XPointer]).
```

Приведем пример. В отрывке из «Гамлета» следующий указатель:

```
Id(a23).span(child(1),child(5))
```

возвратит все дочерние узлы с первого по пятый, то есть:

[первый дочерний узел, тип=элемент]
[второй дочерний узел, тип=текст]

```
<SPEAKER> Polonius </SPEAKER>
```

[третий дочерний узел, тип=элемент]
[четвертый дочерний узел, тип=текст]

```
<DIRECTION> crossing downstage </DIRECTION>
```

[пятый дочерний узел, тип=текст]

```
Fare you well, my lord.
```

Для ясности мы добавили номера и типы дочерних узлов.

Строковый терм места

Спецификация XPointer позволяет выбирать определенные строки и части строк, используя ключевое слово `string`.

Вот общий вид синтаксиса:

```
[абсолютное место].string([целое число: - показывает какой из экземпляров данной
строки использовать | all(все)], [строка, которую нужно найти], [целое число: -
показывает, откуда начать возврат], [целое число: - число символов, которые нужно
возвратить])
```

В этом довольно длинном синтаксисе можно запутаться. По сути дела мы выбираем, какой экземпляр строки искать, какую именно строку искать, с какой части строки начинать возврат, и сколько символов возвращать.

Приведем несколько примеров, которые должны прояснить ситуацию. Вот простой XML-документ, включающий детские стихи:

```
<xdoc>
```

```
    <verse id= "v1">
    Here we go gathering nuts in May,
        nuts in may,
        nuts in may,
    Here we go gathering nuts in May,
        nuts in may,
        nuts in may,
    All on a frosty morning.
    </verse>
</xdoc>
```

Следующая запись:

```
id(v1).string(3, "")
```

указывает на позицию, находящуюся непосредственно перед третьим символом, то есть перед буквой **e**. Напоминаем, что в XML символ перехода на новую строку также считается символом. (На самом деле существуют и табуляторы, которые тоже необходимо учитывать.)

Этот XPointer:

```
id(v1).string (2, "nuts in may")
```

указывает на позицию непосредственно перед вторым выражением `nuts in may`.

Следующий Xpointer:

```
id(v1).string(2, "nuts in may",9)
```

указывает на позицию непосредственно перед словом `may` второго выражения `nuts in may`.

Следующий указатель XPointer:

```
id(v1).string(2, "nuts in may",9,3)
```

идентифицирует позицию, находящуюся непосредственно перед словом `may` второго `nuts in may`, и возвращает три символа, то есть строку `may`.

На этом закончим краткий обзор указателей.

Мы рассмотрели примеры, которые в основном используют только ключевое слово `child`. Использование других ключевых слов по сути практически ничем не отличается. Подробности описаны в спецификации. Изучив этот раздел, вы сможете без особых трудностей в ней разобраться.

Прежде чем завершить обсуждение указателей, скажем несколько слов о синтаксисе указателей XPointer и определении типа документа.

X-указатели и определение типа документа

В приведенных в этой главе примерах использованы только правильные документы, однако в любом состоятельном документе XML все специальные атрибуты также должны быть описаны в определении типа документа.

К счастью, указатели XPointer являются просто частью значения атрибута `href`, поэтому достаточно простого описания значения атрибута `href` как CDATA.

Все другие атрибуты ссылок, которые мы используем в нашем документе, конечно же, обязательно должны быть описаны. Вот пример описания атрибута простой ссылки для элемента `mylink`.

```
<!ATTLIST mylink
xml:link CDATA #FIXED "simple"
   href CDATA #REQUIRED
>
```

Заключение

В этой главе мы рассмотрели язык создания ссылок в XML (XLL) и убедились, насколько шире возможности этих спецификаций по сравнению с простыми ссылками в HTML. Мы обсудили:

- простые ссылки в HTML;
- простые ссылки в XML и их дополнительные возможности по сравнению с простыми ссылками в HTML;
- расширенные ссылки и их использование в качестве встроенных и внешних;
- некоторые требования к программному обеспечению, необходимые для поддержки расширенных ссылок;
- указатели в XML и их возможности по сравнению с простыми идентификаторами фрагментов и рудиментарными указателями в HTML.

Надеемся, теперь, когда эти операции будут поддерживаться браузерами, вам удастся сравнительно легко освоить спецификации на ссылки. Как уже было сказано, недавно была сформирована рабочая группа по разработке ссылок в XML, поэтому вскоре следует ожидать появления рабочих проектов.

Глава 6. Объектная модель документа XML

Одна из самых сложных проблем, возникающих при создании любого стандарта типа XML, состоит в том, что каждая группа программистов, занимающихся разработкой приложений, стремится создать свой собственный метод обработки документа. Этот порок становится особенно очевидным на примере динамического HTML. Для выполнения одного и того же процесса, связанного с практическим применением данного языка, два основных производителя браузеров разработали заметно отличающиеся методы и синтаксис.

Объектная модель XML-документа, предлагаемая консорциумом W3C (The W3C XML Document Object Model), относится к одним из наиболее значимых стандартов всей конфигурации после разве что основополагающей спецификации XML. Причина в том, что объектная модель предоставляет разработчикам универсальный словарь для обработки XML-документа. Независимо от того, написано ли приложение на языках C, Java, Perl, Python или Visual Basic, оно может использовать одни и те же методы решения задачи. Другими словами, несмотря на то, что происходящее «под капотом» может сильно отличаться по целям и областям использования, команды останутся одними и теми же.

В конце концов, педаль торможения должна во время езды срабатывать одинаково, независимо от того, является ли машина грузовиком или легковым автомобилем.

Повторяем, что для клиента не имеет значения, используют ли системы просмотра браузер Netscape Communicator, Microsoft Internet Explorer (IE5) или какой-либо другой. Если все они реализуют одну и ту же объектную модель документа, его скрипт будет работать. Заметим также, что во время подготовки книги браузер IE5 был практически полностью совместим с объектной моделью XML-документа (DOM).

О чем говорится в этой главе

Здесь будет рассмотрена концепция *объектной модели документа* (Document Object Model, DOM) применительно к XML-документам, и в частности, «ядро» модели, объявленное консорциумом W3C, представляющее собой *интерфейс программы-приложения* (Application Program Interface, API) для различных приложений. Вначале перечислим принципиальные положения, на которых мы остановимся особо. Это прежде всего:

- модель дерева XML-документа;
- объектная модель XML-документа;
- значение общепринятого интерфейса прикладной программы (API).
- затем исследуем предлагаемый интерфейс и с помощью браузера IE5 продемонстрируем примеры анализа и построения документов с использованием скриптов ECMAScript.

В этой главе мы ограничимся только *первым уровнем ядра интерфейса API объектной модели документа* (W3C DOM Level 1 Core API), разработанной консорциумом W3C, а также примерами, использующими связывание со скриптами ECMAScript (JavaScript).

Спецификацию объектной модели документа можно найти по адресу http://www.w3.org/TR/REC-DOM-Level-1.

Общее представление о моделях документа

Прежде чем перейти к объектной модели, разработанной консорциумом W3C, рассмотрим принципиальную схему ее построения, а также некоторые из требований, которые участвующие в проекте исследователи предъявляют подобным объектам.

Интересно, известно ли читателям, что все они не раз встречались с идеей объектной модели. Каждый день нам приходится читать и просматривать книги, журналы, разбираться с кипой документов, при этом никто из читателей порой не подозревает, что в таких случаях он пользуется той или иной моделью восприятия и запоминания прочитанного.

Линейная модель

Если бы я попросил вас найти строку 18 на странице 243 в книге «Профессиональные таблицы стилей для HTML и XML», вы без труда выполнили бы мое указание и обнаружили, что находитесь в разделе, озаглавленном «Объектная модель документа».

Я использовал *линейную модель* (linear model) для описания книги, и мог бы и далее требовать от вас поступать подобным образом, попросив вас найти второе слово заголовка и его третью букву. Но при таком подходе мне придется ограничиться символом, как предельной единицей, поскольку символ, как и атом в классической физике, игнорирующей его строение, является мельчайшей неделимой частицей текста.

С помощью некоторого условного языка программирования можно записать:

```
Var myBook= Professional Style Sheets for HTML and XML
X=myBook.page(243).line(18)
Print X
```

Результатом печати была бы строка `"Объектная модель документа"`.

Линейная модель хорошо работает со статическими объектами, например, с книгой, но представьте себе, что произойдет, если книга будет переработана или выпущена в другом формате. Ясно, что прежнее линейное описание перестанет действовать.

Возьмем главу, над которой я сейчас тружусь. Как сослаться на ту или иную страницу или раздел, если книга еще даже не напечатана? Очевидно, требуется другой тип модели; с ним вы тоже знакомы.

Модель дерева документа

Я могу сказать: «Найдите первый абзац пятой главы книги «XML. Новые перспективы WWW».

Для динамических документов эта модель намного удобнее – до тех пор, пока редактор не решит добавить еще пару разделов или изменить порядок следования глав. Чтобы эта модель работала, необходимо иметь оглавление и менять его каждый раз, когда нарушается порядок следования отдельных частей.

Этот вариант широко используется при работе с документами и называется *моделью дерева* (tree model). В этом типе каждая глава или абзац рассматриваются как *узел* (node) дерева. Начиная с любой точки, можно попасть в любую часть дерева, передвигаясь по нему с помощью следующих команд: «Пройти три абзаца назад, затем два предложения вперед».

В биологии узлом называется место ветвления дерева или любого другого растения. Давайте начнем наше путешествие с корневого узла и доберемся до листьев, которые можно сравнить с отдельными символами документа. Каждому узлу соответствует *родительский узел* (parent node) и несколько *дочерних узлов* (child nodes), не считая, разумеется, листьев на концах ветвей. Узлы одной и той же ветви называются *узлами-братьями* (siblings).

Для динамического документа эта рабочая модель существенно удобнее, однако она тоже имеет недостатки: каждый раз, когда в документ вносятся изменения, большую часть дерева нужно рисовать заново.

Объектная модель

Недостатки предыдущих моделей приводят нас к третьему, тоже известному типу – *объектной модели* (object model). В ней каждая часть книги рассматривается как объект. Книга, как и глава или раздел, тоже является объектом со своими свойствами.

Преимущество этой модели состоит в том, что все, что в этом случае следует сделать – это назвать имя объекта, к которому требуется получить доступ. Таким образом, фраза-команда: «Возьмите сборник «XML. Новые перспективы WWW», найдите главу «Объектная модель документа XML», подраздел «Объектная модель», – рано или поздно приведет вас к тому фрагменту, который вы сейчас читаете независимо от того, какие действия производятся над расположением других частей.

Поскольку объекты имеют имена, всегда можно найти соответствующее место, даже в том случае, если редактор решит сократить книгу или поменять порядок глав на обратный.

XML-документы обычно являются динамическими образованиями, поэтому для их построения необходима модель, которая не только описывает части, встроенные в документ, но и может их изменять и дополнять. Объектная модель документа, разработанная консорциумом W3C, обладает всеми указанными функциями. Она объединяет в себе концепцию дерева и концепцию объекта.

Методы сборки

В детстве родители покупали нам сборные модели самолетов или кораблей, к ним прилагались яркие наборы чертежей. Изучать чертежи было очень интересно, однако настоящее, до глубины сердца удовольствие мы получали при сборке модели.

Это справедливо и для моделей XML-документа. Возможность детально описывать документ очень хороша, но нельзя обойтись и без способа его построения. Пользователь должен иметь возможность сказать: «Поместите здесь элемент» или «Добавьте сюда текст» и «Вставьте на эту страницу комментарий».

Группа методов, которые позволяют реализовывать подобные операции, в спецификации объектной модели документа XML известны под общим названием *методов сборки* (factory methods). Хорошая объектная модель документа позволяет собрать сложный документ из отдельных заметок. *Интерфейс приложения объектной модели документа* (W3C DOM API), разработанный консорциумом W3C, обеспечивает широкий набор подобных методов; с ними мы будем знакомиться по мере изучения данной главы.

Дерево XML-документа

Рассмотрим правильный XML-документ и нарисуем соответствующее ему дерево. Документ выглядит следующим образом:

```
<xdoc>
   <greeting>Здравствуй, XML</greeting>
   <farewell>До свидания, HTML</farewell>
</xdoc>
```

Соответствующее ему дерево показано на рис. 6.1.

Этот документ имеет один дочерний узел – так называемый корневой узел, имеющий, в свою очередь, два дочерних узла, `greeting` и `farewell`, которые являются элементами. Каждый из них также имеет текстовый дочерний узел.

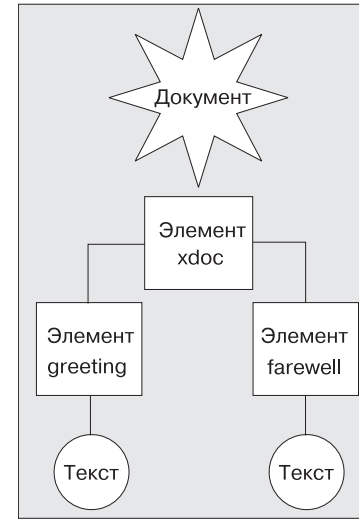


Рис. 6.1. Дерево простого XML-документа

Совет *Конечно, можно сказать, что текстовые узлы имеют дочерние узлы – слова; те в свою очередь тоже имеют дочерние узлы – буквы, но в нашем делении документа мы остановимся на узлах, под которыми будем понимать строки текста (по крайней мере, такой метод деления документа предлагает объектная модель, разработанная консорциумом W3C).*

Давайте немного дополним наш XML-документ:

```
<xdoc>
   <greeting>Здравствуй, XML</greeting>
   <farewell>До свидания, <emph>(ни за что!)</emph>
```

```
 HTML</farewell>
</xdoc>
```

Мы добавили один элемент `emph`, но, как вы можете заметить (рис. 6.2), это привело к возникновению не одного, а нескольких дополнительных узлов. Текстовая строка, в которую элемент был вставлен, расщепилась на два узла. Несмотря на то, что наше дополнение к документу кажется незначительным, структура дерева намного усложнилась. В то же время основные взаимоотншения остались прежними.

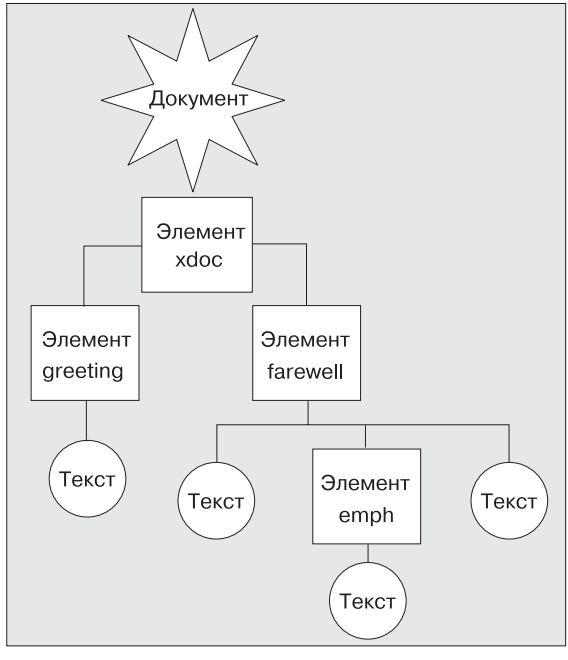


Рис. 6.2. Дерево дополненного XML-документа

Документ XML как совокупность объектов

Давайте вернемся к примеру, на который мы только что ссылались. Зададимся вопросом, сколько в нем объектов, и каковы их свойства?

```
<xdoc>
   <greeting>Здравствуй, XML</greeting>
   <farewell>До свидания, HTML</farewell>
</xdoc>
```

Конечно, мы в состоянии расщепить документ до такой степени, что каждое слово и символ будут иметь свое собственное свойство (код ASCII, например), но в предлагаемом упражнении мы остановимся на текстовых блоках.

Объекты XML

Вот список объектов приведенного выше простого XML-документа:

- объект `Document` – конечно же, включает весь документ;
- объект `xdoc` – это элемент, который имеет два дочерних объекта: элементы `greeting` и `farewell`;
- объект `greeting` – еще один элемент, который имеет элемента-брата – объект `farewell` и дочерний текстовый объект;
- текстовый объект `"Здравствуй, XML"`, дочерний объект элемента `greeting`;
- объект `farewell` – еще один элемент, который имеет элемента-брата (объект `greeting`) и дочерний текстовый объект;
- текстовый объект `"До свидания, HTML"`, дочерний объект элемента `farewell`.

Теперь рассмотрим некоторые свойства, которые следует присвоить каждому объекту.

Возможные свойства

После того, как мы рассмотрели объекты, на которые можно подразделить наш правильный документ, давайте подробнее разберемся со свойствами, которые должен иметь каждый объект. Это необходимо, чтобы построить копию оригинального документа по его модели. Очевидно, что если известно имя каждого объекта, его родителя и его расположение среди объектов-братьев, появляется возможность воссоздать полный документ по его компонентам.

Летописцы английской аристократии использовали эту систему столетиями. Запись «Джон Вельможинс, третий сын пятого герцога Барчестера» очень точно описывает положение Джона в фамильном дереве Барчестеров. Если бы мы имели такую же информацию обо всех членах рода Барчестеров, было бы нетрудно восстановить все дерево.

Распространяя эту аналогию на узлы XML-документов, отметим некоторые свойства, которые можно было бы им присвоить. Несмотря на то, что мы еще не приступали к детальному разбору спецификации объектной модели документа, разработанной консорциумом W3C, укажем главные предлагаемые ими свойства объектов. Очевидно, что не все они необходимы каждому типу узлов. Например, у текстового узла не будет свойства `attribute`, то есть свойство `attribute` для текстового узла всегда имеет значение `null`.

Ниже приведены некоторые возможные свойства узлов:

- *тип узла* (node type). Это свойство показывает, является ли узел комментарием, элементом, командой приложения, участком CDATA, текстовым узлом или чем-либо еще;
- *имя* (name). Именем узла `element` может быть имя его тэга. Для текстового узла именем, предлагаемым спецификацией, разработанной консорциумом W3C, является зарезервированное имя `#text`;
- *значение* (value). В спецификации, разработанной консорциумом W3C, значением узла `element` является `null`, а значением текстового узла – текстовая строка;

- *родитель* (parent). Это свойство показывает, какой узел является родительским для текущего узла. Корневой узел документа родительских узлов не имеет;
- *список дочерних узлов* (child list). Это свойство перечисляет все дочерние узлы данного узла, от первого до последнего;
- *узлы-братья* (siblings). Это свойство указывает на положение узла в списке родственных единиц, или, другими словами, какие узлы являются его старшими и младшими братьями. Несмотря на то, что данное свойство не является необходимым для точного определения расположения узла на дереве, так как можно найти родителя и вывести список его дочерних узлов, – оно, конечно же, является полезным
- *атрибуты* (attributes). Это свойство представляет собой список пар атрибут-значение для данного элемента. В XML порядок следования этих пар не важен. В принципе атрибуты могут рассматриваться и как дочерние узлы. Для объектной модели документа такой подход вполне оправдан. Однако в объектной модели документа DOM, разработанной консорциумом W3C, атрибуты рассматриваются как свойства узла `element`, а не как дочерние узлы.

Типы узловых объектов

Проницательный читатель, конечно, успел заметить, что до сих пор мы рассматривали текстовые узлы и узлы-элементы, но не упоминали о других типах объектов, характерных для XML-документов. К ним относятся комментарии, команды приложений, описания XML и определение типа документа. Очевидно, что для определения типа документа понадобится его собственный полный набор типов узлов.

Мы опустили другие типы узлов потому, что говорили об объектной модели документа в целом. Очевидно, что в полной объектной модели документа способы описания этих узлов должны быть предоставлены, как, например, в объектной модели документа DOM, разработанной консорциумом W3C, где они описаны. С конкретным способом реализации подобного описания мы встретимся чуть позже. Однако следует заметить, что основная спецификация объектной модели документа DOM, разработанной консорциумом W3C, не рассматривает определение типа документа в полном объеме.

Теперь, когда мы получили общее представление об объектной модели документа, а также познакомились с содержащейся в ней информацией, необходимой для анализа и реконструкции XML-документа, исследуем рекомендуемый интерфейс программы-приложения, разработанный консорциумом W3C для объектной модели документа.

Интерфейс приложения для объектной модели документа

Возможно, с появлением объектной модели документа консорциума W3C разработчики Web, в конце концов, получат желаемый результат: стандартный, независимый от производителя интерфейс программы-приложения (API), который

позволит обрабатывать Web-страницы, не беспокоясь об особенностях конкретного браузера или языка скриптов. Действительно ли таковой механизм как объектная модель документа является вехой в развитии Web или это еще один обманчивый огонек в надежде отыскать удобный для всех, работоспособный стандарт? Ниже мы поясним, что означает описываемый здесь механизм для Web-разработчиков, и каковы перспективы его применения. Затем посмотрим, что нас ожидает в будущем.

Значение общепринятого интерфейса приложения

В настоящее время при решении любой, исключая разве что самые тривиальные, задачи с помощью скрипта в HTML-странице очень скоро у пользователя неизбежно начинаются проблемы. Необходимо послать запрос браузеру, чтобы определить, какого он типа, затем только браться за написание различных кодов для выполнения одной и той же задачи, в зависимости от того, с каким браузером – компании Netscape или Microsoft – приходится иметь дело.

Точно так же в том случае, когда программы сформированы на языке C или на языке Visual Basic или даже при наличии двух программ, написанных на одном и том же языке, реализованном различными производителями, – специалистам приходится использовать различные методы для выполнения одной и той же задачи.

Общепринятый интерфейс программы-приложения (API), при условии, что различные производители будут ему следовать, позволит использовать одни и те же команды для выполнения одной и той же задачи в любом приложении.

Насколько проще тогда будет жизнь бедного программиста!

Похоже, с помощью языка XML мы обрели эту чашу Грааля, потому что основные производители обещали придерживаться интерфейса объектной модели документа, разработанной консорциумом W3C, и в браузере IE5 уже реализована его основная часть.

Прежде чем рассматривать специфику объектной модели документа, предложенной консорциумом W3C, обратимся к интерфейсу, который использует данная объектная модель, а именно, интерфейсу, соответствующему *архитектуре общих брокеров объектных запросов* (Common Object Request Broker Architecture, CORBA).

Язык определения интерфейсов группы управления объектами

Перед тем как перейти к изложению означенной темы, коротко обсудим синтаксис, который используется для определения свойств и методов в спецификации объектной модели документа, разработанной консорциумом W3C. Для многих, из-за отсутствия навыка мыслить в терминах распределенных объектов, эта спецификация недостаточно понятна. У специалистов по языку Java, знакомых с *активизацией удаленного метода* (Java's Remote Method Invocation, RMI), не возникнет проблем с терминологией, и они могут пропустить этот раздел.

Совет *Те из читателей, кому интересно узнать побольше об активизации удаленного метода (RMI), могут прочитать книгу Beginning Java («Начинаем изучать Java»), написанную Айвором Хортоном (Ivor Horton) и выпущенную издательством Wrox Press, ISBN 186002238.*

Язык определения интерфейсов группы управления объектами (Object Management Group's Interface Definition Language, OMG IDL) является механизмом, который позволяет приложениям и их объектам общаться друг с другом, несмотря на то, что они написаны на разных языках программирования.

Большинство читателей знакомо с моделью клиент-сервер. Если при использовании браузера я активизирую ссылку, то мой браузер (клиент), посылает запрос серверу, чтобы тот отыскал нужный документ, то есть объект. В любой сети машина, делающая запрос, является клиентом, а машина, предоставляющая услуги, – сервером.

Язык OMG IDL создан для определения интерфейсов объектов. Если существует программа, написанная на языке Java, которая должна по сети установить связь с программой, написанной на языке C++, эту операцию можно выполнить, используя соответствующий брокер объектных запросов; он как раз будет служить переводчиком между программами. Брокер конструируется в соответствии с архитектурой CORBA, определяемой группой управления объектами OMG с использованием языка IDL.

Аналогия

Возможно, весь этот материал проще понять, если прибегнуть к аналогии.

Предположим, у меня есть инвестиционный портфель с ценными бумагами и облигациями (хотел бы я, чтобы это действительно так и было!). У меня могут быть государственные облигации, муниципальные облигации, акции Лондонской Фондовой биржи, Нью-Йоркской Фондовой биржи (NYSE) и Токийской Фондовой биржи.

Если бы я испытывал желание купить акции Microsoft на Нью-Йоркской Фондовой бирже, передо мной встал бы выбор: сделать это самому или обратиться к брокеру.

В том случае, если я приму решение заняться этим самостоятельно, мне прежде всего придется купить место на Нью-Йоркской Фондовой бирже, что может обойтись мне в несколько миллионов долларов! Затем пришлось бы изучить все правила, необходимые для совершения транзакции, сделать транзакцию и заполнить все бумаги, необходимые для ее завершения. А если бы мне у меня появилось желание заняться той же самой операцией в Японии, мне пришлось бы научиться говорить по-японски!

Куда проще и целесообразнее сразу обратиться к брокеру и попросить его купить акции.

Однако прежде чем воспользоваться услугами моего брокера, я должен обсудить набор протоколов; другими словами, согласовать условия, на которых мы займемся нашим бизнесом. Следующий шаг, необходимый при заключении сделки – это утверждение процедуры нашего общения, или, в компьютерных терминах, интерфейса.

Теперь для связи с брокером я буду пользоваться этим интерфейсом, который в нашем случае представлен самой обычной формой (это может быть телефонный разговор или Web-сайт).

Первый интерфейс, который мне даст брокер – это интерфейс инвестиционного портфеля, имеющий некоторые атрибуты, которые приведут меня к другому интерфейсу и методам, позволяющим завершить дело.

Атрибутами могут быть облигации, ценные бумаги и так далее, они отправят меня далее, к интерфейсам облигаций, ценных бумаг и т. д.

Методы, в свою очередь, могут включать метод `ShowInvestmentList()` (показать список инвестиций), который выдаст мне список моих инвестиций.

Если бы я запросил интерфейс для ценных бумаг, он мог бы иметь такие методы как `GetShareList()` (получить список акций) или `GetSharePrice([название акции])` (получить цену акций), или `SellShare([название акции],[количество])` (продать акции) и так далее.

Идея состоит в том, чтобы мне уже не надо беспокоиться о деталях любой из транзакций, их проведение или разработку я, используя указанные методы, предоставил бы брокеру. Теперь, при желании провести ту или иную операцию, я должен связаться по подходящему интерфейсу и отдать соответствующие команды.

Синтаксис IDL

Язык OMG IDL работает на основе тех же принципов. Вместо того чтобы беспокоиться о типе документа или о языке, на котором написан процессор, я могу использовать определения языка IDL, а *брокер объектных запросов* (Object Request Broker, ORB) сделает за меня всю работу.

В компьютерных терминах получение доступа к документу означает получение доступа к объекту документа, содержащему узловые объекты, которые могут быть элементами, комментариями и тому подобными понятиями.

Однако в терминах синтаксиса IDL я использую интерфейс между клиентом и этими объектами, и направляю все мои запросы и методы интерфейсу ORB, который передает запрос.

Чтобы не было путаницы между полями, которые должен иметь интерфейс, и атрибутами элементов XML, спецификация различает `attribute` (атрибут) интерфейса и узлы интерфейса `Attr`, представляющие атрибуты XML.

Интерфейс `Document` имеет атрибут `documentElement`, поэтому используя синтаксис:

```
X=Document.documentElement
```

можно загрузить в переменную `X` узловой объект, который является корневым элементом документа. Сам объект может быть написан на языке C++, я могу также воспользоваться языком Java, однако интерфейс гарантирует, что доступ к свойствам и их изменение можно осуществлять, используя запросы на любом языке.

Статус объектной модели документа

Объектная модель документа (DOM) в близком будущем вполне может оказаться одним из наиболее важных документов Internet, которые, правда, еще предстоит создать. Поэтому имеет смысл рассмотреть уже решенные на сегодняшний день задачи рабочей группы по DOM консорциума W3C, а также те проблемы, которыми предполагается заняться в будущем.

Первоначальный вариант DOM представляет собой минимальный набор объектов и интерфейсов для доступа к документам XML (и другим) и для их обработ-

ки. Этот механизм не позволяет создавать или сохранять XML-документ. Такие функции выполняет и несет за них ответственность агент пользователя.

Несмотря на то, что основная объектная модель прекрасно подходит для обработки документов, к сожалению, она не охватывает все стороны этой темы. Они будут реализованы во втором уровне DOM.

В текущей (Level I) DOM не рассматриваются:

- структурная модель для внутреннего и внешнего подмножеств DTD или другой схемы;
- проверка на состоятельность в соответствии с этой схемой;
- контроль над воспроизведением документов с помощью таблиц стилей.

Эти вопросы, повторяем, будут рассмотрены во втором уровне (Level II) DOM.

Интерфейсы объектной модели документа

Объектная модель представляет документы в виде иерархии узловых объектов, создающих интерфейсы. Ниже приведен список интерфейсов XML-узлов, которые поддерживает объектная модель DOM. Для каждого указаны его дочерние узлы, которые поддерживает соответствующий интерфейс. Кроме того, приведено имя и значение узла каждого типа, а также краткое пояснение его свойств. Более детальный анализ приведен позже в разделе «Примеры интерфейсов объектной модели документа».

Сейчас заметим только, что каждый интерфейс представляет собой объект, которому можно присваивать следующие свойства:

- *тип* (type), как мы увидим позже, определяется целым числом или строковой константой. Все интерфейсы имеют какое-либо значение свойства `type`. Свойство узла: имя;
- *имя* (name) может быть разным для каждого конкретного объекта. Например, именем элемента является имя его тэга. В то же время объект может иметь обобщенное имя. Например, узел `Comment` (комментарий) всегда имеет имя `#comment`, а узел `Document` (документ) – имя `#document`. Перед обобщенным именем всегда употребляется значок `#`;
- *значение* (value). Многие интерфейсы не имеют значения. В этом случае оно всегда равно `null`. В других случаях значение определяется содержанием узла, например, значением текстового узла является строка, которую он содержит;
- *атрибуты* (attributes). Единственным интерфейсом, который возвращает значение для свойства `attributes`, является интерфейс `Element`.

Познакомимся с типами интерфейсов, которые поддерживает объектная модель документа.

Интерфейс Document

Узел `Document` (документ) содержит следующие дочерние узлы: `Element` (не более чем один), `ProcessingInstruction` (команда приложения), `Comment` (комментарий), `DocumentType` (тип документа).

Разумеется, документ всегда представляет собой единое целое. Методы создания документа не предоставляются, эта задача возлагается на агента пользователя. Документ может иметь минимальный размер. Например, документ вида `<xroot></xroot>` является правильным и может быть использован как основа для построения более сложного документа.

Напоминаем, что объектная модель документа не предоставляет способа сохранения сформированного документа, эта задача отводится агенту пользователя. Позже будет показано, что большинство документов, построенных с помощью скрипта, исчезает при закрытии окна браузера клиента вследствие того, что скрипты клиентов (например, ECMAScript), используемые в браузерах, не имеют доступа к жесткому диску сервера.

Значение свойства `Name` интерфейса `Document` равно `#document`, а значение свойства `Value` интерфейса `Document` равно `null`.

Совет *Несомненно, читателям известно, что значение null не совпадает со значением 0 (ноль). «Ноль» показывает, что некоторое значение существовало или может существовать; null – что значение никогда не присваивалось. Когда интерфейс объектной модели документа имеет значение null, предполагается, что ему и не должно присваиваться никакое значение.*

Документ может иметь только один узел `Element`, а именно корневой узел. На схеме (рис. 6.3) показан узел `Document` и типы дочерних узлов, которые он может иметь.

Интерфейс DocumentFragment

Узел `DocumentFragment` (фрагмент документа) содержит следующие дочерние узлы: `Element` (элемент), `ProcessingInstruction` (команда приложения), `Comment` (комментарий), `Text` (текст), `CDATASetion` (секция `CDATA`), `EntityReference` (ссылка на компонент).

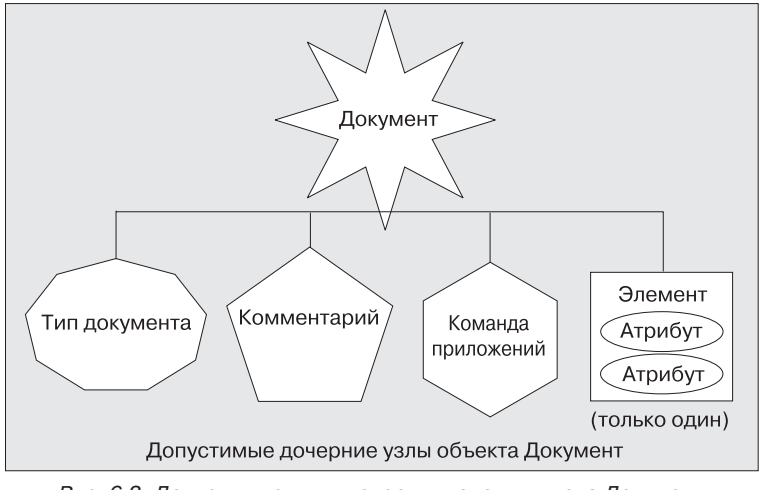


Рис. 6.3. Допустимые типы дочерних узлов для узла Документ

Фрагмент документа удобен в применении. Мы можем создать его, заполнить различными узлами, а затем вставить в документ. Необходимо только поместить включаемый отрывок таким образом, чтобы основной документ остался правильным.

Значение свойства `Name` интерфейса `DocumentFragment` равно `#document-fragment`, а значение свойства `Value` интерфейса `DocumentFragment` равно `null`.

Интерфейс DocumentType

Узел `DocumentType` (тип документа) содержит следующие дочерние узлы: `Notation` (обозначение), `Entity` (компонент).

Объектная модель документа, разработанная консорциумом W3C, не поддерживает полное определение типа документа (DTD). Интерфейс `DocumentType` предоставляет доступ к компонентам и обозначениям, содержащимся в DTD. В то же время в основной объектной модели документа первого уровня (Level I Core DOM) описания элементов и атрибутов не поддерживаются.

Именем интерфейса `DocumentType` является строка, которая следует сразу за словом `<!DOCTYPE`. Например, в декларации `<!DOCTYPE xdoc SYSTEM="somedoc.dtd">` значением свойства `Name` интерфейса `DocumentType` является `xdoc`. Разумеется, это имя совпадает с именем корневого элемента Значение свойства `Value` интерфейса `DocumentType` равно `null`.

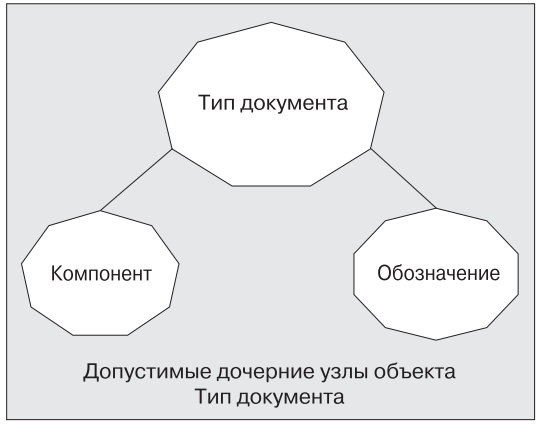


Рис. 6.4. Допустимые типы дочерних узлов для узла Тип документа

Интерфейс EntityReference

Узел `EntityReference` (ссылка на компонент) содержит следующие дочерние узлы: `Element` (элемент), `ProcessingInstruction` (команда приложения), `Comment` (комментарий), `Text` (текст), `CDATASection` (секция `CDATA`), `EntityReference` (ссылка на компонент).

Дочерним узлом для узла `EntityReference` может быть целый фрагмент документа, а также другие узлы `EntityReference` (рис. 6.5).

Значением свойства `Name` интерфейса `EntityReference` является имя компонен-

та, на который дана ссылка. Например, значением свойства `Name` ссылки на компонент `&dom` является имя `dm`. Значение свойства `Value` интерфейса `EntityReference` равно `null`.

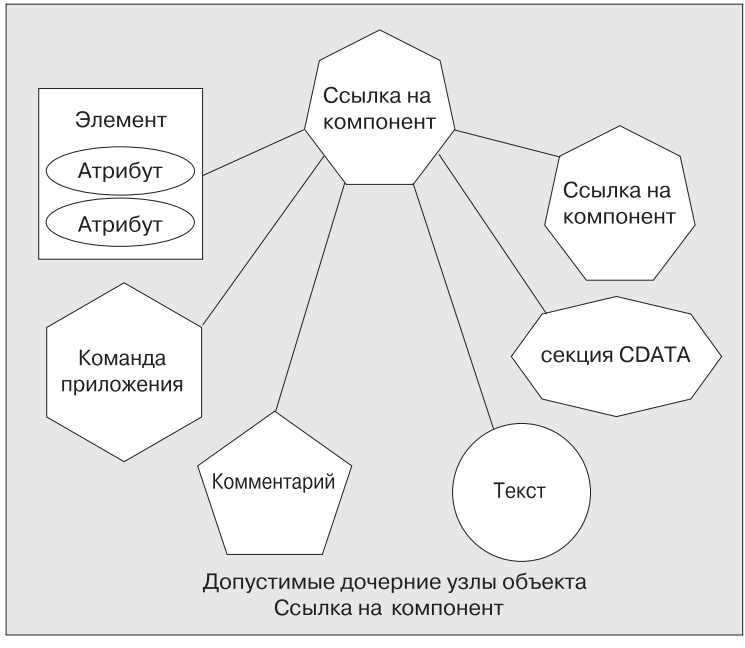


Рис. 6.5. Допустимые типы дочерних узлов для узла Ссылка на компонент

Интерфейс Element

Узел `Element` (элемент) содержит следующие дочерние узлы: `Element`, `Text` (текст), `Comment` (комментарий), `ProcessingInstruction` (команда приложения), `CDATASection` (секция CDATA), `EntityReference` (ссылка на компонент).

Большинство узлов в обычном документе представляет собой текст и элементы. Атрибуты элемента не относятся к его дочерними узлами, а рассматриваются как свойства узла `Element`. Более полную информацию можно найти в разделе «Интерфейс Attr».

Значением свойства `Name` элемента является имя его тэга, а значение свойства `Value` интерфейса `Element` равно `null`. Интерфейс `Element` имеет также свойство `attributes`, которое содержит объект `NameNodeMap` (карта_имен_узла). Этот объект будет рассмотрен позже при обсуждении доступа к парам атрибут-значение с помощью кода.

Дочерние узлы элемента показаны на рис. 6.6.

Интерфейс Attr

Узел `Attr` (атрибут) содержит следуюие дочерние узлы: `Text` (текст), `EntityReference` (ссылка на компонент).

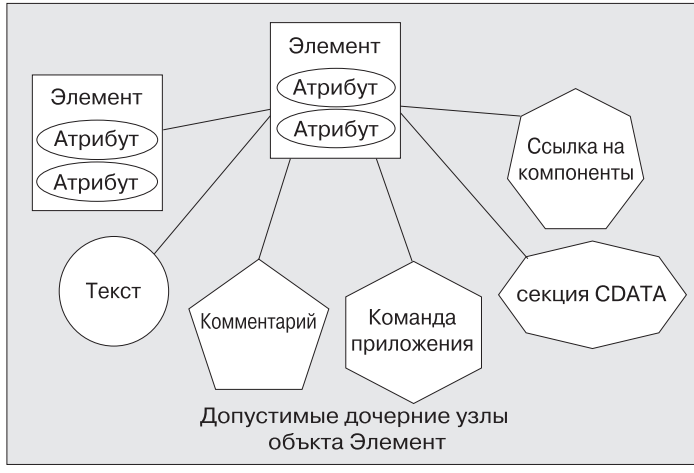


Рис. 6.6. Допустимые типы дочерних узлов для узла Элемент

В объектной модели документа имя интерфейса `attribute` сокращено до `Attr`, чтобы не путать его с ключевым словом `attribute` интерфейса IDL. Узлы `Attr` не рассматриваются как дочерние узлы элемента в том смысле, что существует самый старший и самый младший брат. Порядок, в котором появляются узлы `Attr`, не является существенным. Пару атрибут-значение лучше всего представить себе как свойство элемента, содержащееся в узле `Attr`.

Чтобы избежать недоразумений, заметим, что существуют следующие обозначения:

- `Attr` – тип узла DOM, представляющий атрибут элемента XML;
- `attributes` – конкретное поле интерфейса типа `NamedNodeMap` (см. позже);
- интерфейсы, определенные с использованием терминологии IDL, имеют свойства (поля) типа `attribute` (без 's').

Таким образом, `attributes`, представляет собой атрибут интерфейса IDL типа `NameNodeMap`.

В XML ссылка на внутренний компонент представляет собой допустимое значение атрибута (рис. 6.7).

Интерфейс ProcessingInstruction

Не имеет дочерних узлов.

Значением свойства `Name` интерфейса `ProcessingInstruction` (команда приложения) является имя приложения, которому адресована команда. Значением свойства `Value` интерфейса является полное содержание команды, то есть все, что находится между символами `<?` и `?>`.

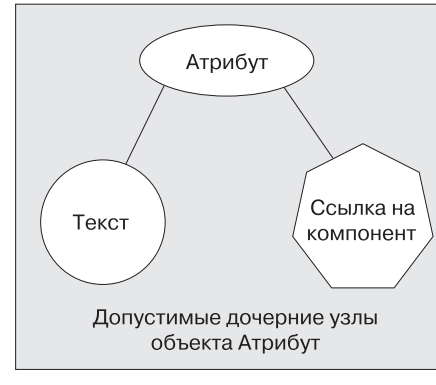


Рис. 6.7. Допустимые типы дочерних узлов для узла Атрибут

Интерфейс Comment

Не имеет дочерних узлов.

Значением свойства `Name` интерфейса `Comment` (комментарий) является `#comment`, а значением свойства `Value` интерфейса `Comment` является его содержание, то есть все, что находится между символами `<!--` и `-->`.

Интерфейс Text

Не имеет дочерних узлов.

Значением свойства `Name` интерфейса `Text` (текст) является `#text`, а значением свойства `Value` интерфейса является содержание его строки текста.

Интерфейс CDATASection

Не имеет дочерних узлов.

Значением свойства `Value` интерфейса `CDATASection` (секция CDATA) является содержание секции CDATA, а значением его свойства `Name` – `#cdata-section`.

Интерфейс Entity

Не имеет дочерних узлов.

Значением свойства `Name` интерфейса `Entity` (компонент) является имя компонента. Значения для свойства `Value` интерфейс `Entity` не имеет.

Интерфейс Notation

Не имеет дочерних узлов (рис. 6.8).

Значением свойства `Name` интерфейса `Notation` (обоначение) является `NOTATION`, а значения для свойства `Value` этот интерфейс не имеет.

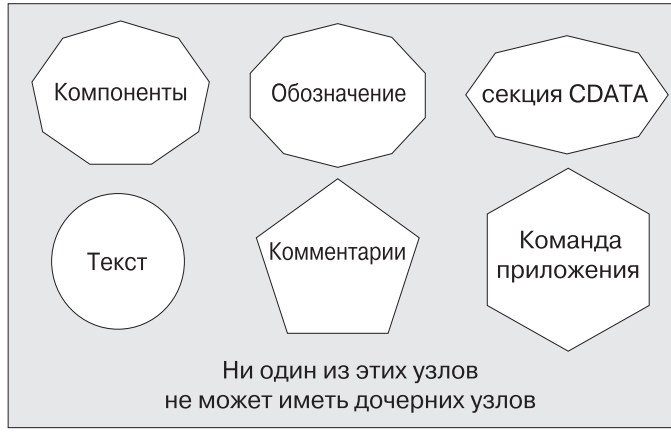


Рис. 6.8. Узлы Компонент, Обозначение, Секция CDATA, Текст, Комментарий, Команда приложения не могут иметь дочерних узлов

Эти краткие описания типов узлов DOM по мере дальнейшего изучения будут развернуты и приобретут конкретный смысл.

Следует учесть, что во время подготовки сборника единственным приложением, реализующим DOM, являлась бета-версия браузера IE5. Поэтому все практические примеры объектной модели будут строиться на ее основе.

Надеемся, что ко времени выхода книги из печати появятся и другие реализации объектной модели документа.

XML в браузере IE5

Поскольку для иллюстрации примеров мы будем использовать браузер IE5, следует обсудить, как XML-документ создается в этом браузере. Существуют два основных метода создания XML-документов: с помощью *островка XML* (XML Island) и *управляющего элемента ActiveX* (ActiveX Control).

Островок XML

Самый простой способ создать объект XML-документа в браузере IE5 – это использовать тэг `<XML>`. Представьте себе, что мы создали очень простой XML-файл, называемый `myxmlfile.xml` и содержащий: `<greeting>Здравствуй, мир!</greeting>`. Следующий код использует идентификатор `ID` и атрибут `SRC` тэга `<XML>` для создания объекта XML-документа:

```
<XML ID="island", SRC="myxmlfile.xml"></XML>
```

XML-документ обязан быть правильным, а если в нем присутствует DTD, то еще и состоятельным, в противном случае анализатор MSXML откажется его загружать.

Имеет смысл, используя метод `parseError`, проверить документ на состоятельность. Заметим, что метод `parseError` является частью браузера IE5, а не объектной модели документа. В приведенный ниже документ мы включили метод `parseError` в виде условного оператора. В результате программа просмотра получит информацию о том, по какой причине документ не загружается.

Совет *Один из основных «коммерческих аргументов» в пользу XML состоит в том, что этот язык требует синтаксической правильности. Следовательно, приложение или агент пользователя могут быть уверены, что получают правильный документ.*

Вот как выглядит код:

```
<HTML>
<XML ID="island", SRC="myxmlfile.xml"></XML>
<SCRIPT>
   myDoc=island;
   if(myDoc.parseError.reason!="")
       {
       alert(myDoc.parseError.reason);
       }
   alert(myDoc.documentElement.nodeName);
</SCRIPT>
</HTML>
```

При желании, мы можем следующим образом поместить наш XML-файл между тэгами:

```
<HTML>
```

```
<XML ID="island">
<Greeting>Здравствуй, XML в IE5</greeting>
</XML>
<SCRIPT>
   myDoc=island;
   if(myDoc.parseError.reason!="")
       {
       alert(myDoc.parseError.reason);
       }
   alert(myDoc.documentElement.nodeName);
</SCRIPT>
</HTML>
```

Если наш XML окажется правильным, он будет загружен как объект `document`. Ниже показано, какое изображение появится на экране после запуска последнего из приведенных файлов (рис. 6.9).

Рис. 6.9. Просмотр HTML-документа с XML-вставкой в браузере IE5

Итак, наши начальный и конечный тэги не совпадают, поскольку мы не позаботились о заглавных и строчных буквах. Переменим имя элемента `Greeting` на `greeting`, и все будет в порядке.

Для уверенности в том, что файл загружен, в код добавлена строка:

```
alert(myDoc.documentElement.nodeName);
```

Потом мы рассмотрим эту запись подробнее, а пока обратите внимание на то, что метод `documentElement.nodeName` должен возвращать имя корневого элемента, то есть строку `greeting`.

Что он и делает (рис. 6.10).

Рис. 6.10. Просмотр исправленного документа в браузере IE5

Элемент ActiveX для XML

Входящий в состав браузера IE5 управляющий элемент ActiveX (ActiveX Control) тоже можно использовать для создания объекта XML-документа.

Вот код, который позволяет это сделать:

```
<HTML>
<SCRIPT>
   // Создаем объект activeX и присваиваем его переменной myDocument.
   var myDocument = new ActiveXObject("microsoft.XMLDOM");
```

```
    // Загружаем файл xml в объект activeX, именуемый myDocument.
    myDocument.load("myxmlfile.xml");
        if(myDocument.parseError.reason !="")
            {
            alert(myDocument.parseError.reason);
            }
        alert(myDocument.documentElement.nodeName);
</SCRIPT>
</HTML>
```

Мы снова провели элементарную проверку на наличие ошибок и убедились, что файл действительно загружен.

На практике не имеет значения, какой метод используется для загрузки XML-документа, рассматриваемого как объект. Автор отдает предпочтение методу XML Island, именно этот механизм используется в примерах, приведенных ниже.

Проверив, как наш XML-документ загружается в браузер, обратимся к некоторым примерам использования объектной модели документа, утвержденной консорциумом W3C на основе браузера IE5.

Примеры интерфейсов объектной модели документа

В этом разделе мы расскажем, какие существуют средства обработки документа. Значения интерфейсов мы пока обсуждать не будем. Наша цель –демонстрация существующих методов, а не создание документов с их помощью. Этот процесс читатели могут довообразить сами.

В заключение, однако, мы приведем два практических примера.

Интерфейсы Document и Node

Эти интерфейсы будут рассмотрены совместно, поскольку они составляют основу модели дерева DOM.

Приводимые ниже примеры не отражают всю совокупность методов и атрибутов, применяемых для обработки документа, однако материала, собранного в этом разделе, вполне достаточно, чтобы разобраться в том, как обрабатывать XML-документ. Особое внимание будет уделено тем методам или атрибутам, которые могут показаться особенно трудными для понимания.

Ниже приведен XML-документ, с которым мы детально познакомимся в процессе изучения интерфейсов `Document` (документ) и `Node` (узел).

```
<?xml version="1.0"?>
   <!DOCTYPE xdoc [
   <!ELEMENT xdoc (greeting)*>
   <!ELEMENT greeting (#PCDATA)>
   <!ENTITY dom "объектная модель документа">
   <!ATTLIST greeting
       type CDATA #IMPLIED
       position CDATA #IMPLIED
   >
   ]>
   <!-- Первый комментарий. -->
```

```
<xdoc>
   <greeting type="cordial" position="first">&amp;amp; Здравствуй, &dom;</greeting>
   <greeting> Здравствуй, XML</greeting>
   <!-- Комментарий. -->
   <?pi сделай это и вот это ?>
</xdoc>
```

Далее показано, каким образом можно включить его в HTML-документ. Обратите внимание, как с помощью синтаксиса `myDoc=xisle;` создается объект Document. Как только объект Document сформирован, он может быть обработан методами DOM.

На этом этапе советуем набрать в текстовом редакторе и сохранить в файле `DOMmain.htm` следующий код HTML:

```
<HTML>
<P>Ниже вы видите участок XML</P>
<XML ID="xisle">
   <?xml version="1.0"?>
   <!DOCTYPE xdoc [
   <!ELEMENT xdoc (greeting)*>
   <!ELEMENT greeting (#PCDATA)>
   <!ENTITY dom "объектная модель документа">
   <!ATTLIST greeting
       type CDATA #IMPLIED
       position CDATA #IMPLIED
   >
   ]>
   <!-- Первый комментарий. -->
 <xdoc>
   <greeting type="cordial" position="first">&amp;amp; Здравствуй, &dom;</greeting>
   <greeting> Здравствуй, XML</greeting>
   <!-- Комментарий. -->
   <?pi сделай это и вот это ?>
 </xdoc>
</XML>
<SCRIPT>
   // Создает объект документа XML.
   var myDoc=xisle;
   var rootEl=myDoc.documentElement;
</SCRIPT>
</HTML>
```

Фрагменты кода, которые вам встретятся во время дальнейшего изложения методов и атрибутов интерфейсов `Document` и `Node`, представляют собой строки кода, включаемые в раздел `<SCRIPT>` файла `DOMmain.htm` для демонстрации управления свойствами XML-документа `myDoc`. По мере изучения мы будем постоянно ссылаться на приведенный код.

Интерфейс Node

Все типы узлов объектной модели документа наследуют основной набор свойств от интерфейса `Node`. Мы разделим свойства интерфейса `Node` на атрибуты и методы сборки.

Атрибуты интерфейса Node, доступные только для чтения

В табл. 6.1 приведен список атрибутов интерфейса `Node`, доступных только для чтения.

Таблица 6.1. Список атрибутов интерфейса Node, доступных только для чтения

Атрибут	Тип	Комментарий
nodeName (имя узла)	строка	Имя узла изменяется вместе с его типом. Именем элемента является имя его тэга, именем текстового узла является #text и так далее. Таблица, приведенная в разделе «Краткий обзор имен и значений типов узлов», показывает имена и значения различных типов узлов
nodeValue (значение узла)	строка	Во многих случаях, например в случае элемента, возвращаемое значение равно null. Повторяем, что в таблице раздела «Краткий обзор имен и значений типов узлов» приведены значения различных узлов
nodeType (тип узла)	целое без знака типа short	В таблице раздела, посвященном атрибуту nodeType, приведены значения, возвращаемые для каждого типа узла
parentNode (родительский узел)	объект Node	Возвращает null, если нет родительского элемента, то есть если это узел Document
childNodes (дочерние узлы)	объект NodeList (список узлов)	Возвращаемый список узлов представляет собой индексированный список дочерних узлов
firstChild (первый дочерний узел)	объект Node	Возвращает null, если нет первого дочернего узла
lastChild (последний дочерний узел)	объект Node	Возвращает null, если нет последнего дочернего узла
previousSibling (предшествующий брат)	объект Node	Возвращает null, если нет предшествующего узла-брата
nextSibling (следующий брат)	объект Node	Возвращает null, если нет следующего за ним узла-брата
attributes (атрибуты)	NameNodeMap (карта имен узла)	Очевидно, что только узел Element возвращает значение NameNodeMap; все другие типы узлов возвращают null. Несмотря на то, что объект NameNodeMap (смотрите ниже раздел «Интерфейс NameNodeMap») имеет индекс, порядок следования не имеет значения (в отличие от списка NodeList) и служит просто для удобства доступа к схеме. Список обычно обрабатывается с использованием имени атрибута в качестве аргумента
ownerDocument (документ-владелец)	узел Document	Каждый узел должен принадлежать документу. Это свойство используется приложениями, в которых одновременно обрабатывается несколько документов

Интерфейс NodeList

Доступ к узлам `NodeList` можно получить с помощью метода `item`, который в качестве аргумента использует `index`. Длину списка можно определить методом `length`. Мы увидим его в действии уже в следующем разделе после рассмотрения атрибута `childNodes`. Интерфейс `NodeList` хранит запись дочерних узлов конкретного узла и их позиций. Он является основой структуры дерева документа в памяти компьютера.

Атрибут childNodes

Атрибут `childNodes` возвращает объект `NodeList`, содержащий все дочерние узлы любого данного узла.

В нашем файле `DOMmain.htm` использование строки

```
var x=myDoc.childNodes;
```

в разделе `<SCRIPT>` присвоит список `NodeList` всех дочерних узлов объекта `myDoc` переменной `x`.

В результате добавления кода

```
var x=myDoc.childNodes;
   var z=x.length;
for(var i=0; i < z; i++)
       {
       document.write(x(i).nodeType+ " | ");
       document.write(x(I).nodeName+ " | ");
       document.write(x(I).nodeValue + "<BR>");
           }
```

на экране появится изображение, аналогичное рис. 6.11.

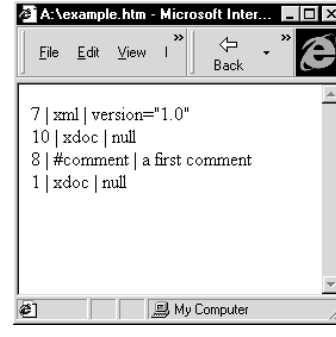


Рис. 6.11. Просмотр файла DOMmain.htm

В следующих разделах после рассмотрения атрибутов `nodeType`, `nodeName`, `nodeValue` будет объяснено, что эти величины означают. Сейчас заметим только, что мы создали переменную `x` типа `NodeList`, содержащую свойства четырех объектов, которые мы смогли получить с помощью индекса.

Атрибут nodeType

Атрибут `nodeType` интерфейса `Node` возвращает целое число, которое указывает на тип узла, с которым мы имеем дело. В табл. 6.2 приведен список различных типов узлов с определяющими их константами и целыми значениями.

Таблица 6.2. Типы узлов с определяющими их константами и целыми значениями

Целое значение	Определяющая константа
1	ELEMENT_NODE
2	ATTRIBUTE_NODE
3	TEXT_NODE
4	CDATA_SECTION_NODE
5	ENTITY_REFERENCE_NODE
6	ENTITY_NODE
7	PROCESSING_INSTRUCTION_NODE
8	COMMENT_NODE
9	DOCUMENT_NODE
10	DOCUMENT_TYPE_NODE
11	DOCUMENT_FRAGMENT_NODE
12	NOTATION_NODE

После того, как к нашему коду был применен атрибут `childNodes`, можно сделать вывод, что узел `myDoc` имеет четыре дочерних узла. Целые числа в начале строки сообщают, что первый из них является описанием XML в форме команды приложения, второй – узлом типа `DOCUMENT_TYPE`, третий узел представляет собой комментарий, а четвертый – элемент.

Атрибуты nodeName и nodeValue

Каждый тип узла обладает именем. Некоторые узлы, кроме имени, имеют и значение. К значениям можно получить доступ, используя атрибуты `nodeName` и `nodeValue` интерфейса `Node`.

Если мы запрашиваем значение, которого нет в узле, то возвращается `null`. Табл. 6.3 в следующем разделе, озаглавленном «Краткий обзор имен и значений различных типов узлов», демонстрирует различные типы узлов, их имена и значения. Значения атрибутов `nodeName`, `nodeValue` и `attributes` изменяются в соответствии с типом узла.

Краткий обзор имен и значений различных типов узлов

Значения атрибутов `nodeName`, `nodeValue` и `attributes` (заметим, что это поле интерфейса, а не `attribute` – тип данных IDL) изменяются в соответствии с типом узла (см. табл. 6.3).

Таблица 6.3. Значения атрибутов в соответствии с типом узла

Тип узла (Node Type)	имя узла (nodeName)	значение узла (nodeValue)	атрибуты (attributes)
Element	имя тэга	null	NameNodeMap
Attr	имя атрибута	значение атрибута	null
Text	#text	содержание текстового узла	null
CDATASection	#cdata-section	содержание участка CDATA	null
EntityReference	имя объекта, на который ссылаются	null	null
Entity	имя объекта	null	null
Processing Instruction	имя приложения, которому адресована команда	все содержание, за исключением имени приложения	null
Comment	#comment	содержание комментария	null
Document	#document	null	null
DocumentType	имя документа	null	null
DocumentFragment	#document-fragment	null	null
Notation	имя обозначения	null	null

Атрибуты родственных отношений

Все перечисленные ниже атрибуты интерфейса `Node` хранят соответствующий узел.

```
parentNode
firstChild
lastChild
```

```
previousSibling
nextSibling
```

Например, в файле DOMmain.htm, приведенном ранее, можно использовать атрибут documentElement, чтобы присвоить корневой элемент документа элементу rootEl:

```
rootEl=myDoc.documentElement;
```

Этот код создаст объект rootEl, который содержит все свойства объекта xdoc. Таким образом, код:

- alert(rootEl.parentNode.nodeName);
  отображает на экране #document;
- alert(rootEl.firstChild.nodeName);
  отображает на экране greeting;
- alert(rootEl.lastChild.nodeName);
  отображает на экране pi (имя приложения, которому адресована команда);
- alert(rootEl.previousSibling.nodeName);
  отображает на экране #comment;
- alert(rootEl.nextSibling.nodeName);
  не отображает на экране ничего, поскольку следующего узла-брата у объекта rootEl не существует.

Атрибут attributes

Атрибут attributes применяется только в узле типа Element. Этот атрибут хранит объект NameNodeMap, содержащий атрибуты данного элемента XML.

В нашем примере мы можем присвоить узел первого элемента greeting переменной anEl с помощью следующего кода в разделе <SCRIPT> файла DOMmain.htm:

```
var rootEl=myDoc.documentElement;
var anEl=rootEl.first.Child;
```

Теперь можно создать объект NamedNodeMap для элемента greeting.

Интерфейс NamedNodeMap

Интерфейс NamedNodeMap имеет методы и атрибуты, приведенные в табл. 6.4. Поместим в раздел SCRIPT нашего файла DOMmain.htm следующий код:

```
var rootEl=myDoc.documentElement;
var anEl=rootEl.first.Child;
```

Напоминаем, что свойство attributes интерфейса Node возвращает интерфейс NameNodeMap. Следующие фрагменты кода иллюстрируют способы, с помощью которых существование объекта NamedNodeMap для узла anEl позволяет получить доступ к различным свойствам:

```
alert(anEl.attributes.getNamedItem("position").nodeValue);
alert(anEl.attributes(1).nodeValue);
alert(anEl.attributes.item(1).nodeValue);
```

Все эти строки кода возвращают значение атрибута `position` первого элемента `greeting`. Это значение равно `first`.

Таблица 6.4. Методы и атрибуты интерфейса NameNodeMap

Метод или атрибут	Допустимые параметры	Комментарий
getNamedItem (получить предмет с заданным именем)	Имя атрибута, доступ к которому нужно получить	Возвращает узел Attr
setNamedItem (установить предмет с заданным именем)	Узел, который следует добавить к объекту NameNodeMap	Не реализован в ранних бета-версиях браузера IE5
removeNamedItem (удалить предмет с заданным именем)	Имя атрибута, который нужно удалить	Если узел Attr, который требуется удалить, является обязательным атрибутом, возникнет ошибка. Если узел, который требуется удалить, является узлом Attr со значением по умолчанию, то он сразу же замещается
item (один из предметов списка)	Целое значение index	Возвращает узел Attr, соответствующий заданному значению index. Если на указанном месте нет атрибута, возвращает null
length (длина) – атрибут	Нет параметров	Хранит длину объекта NameNodeMap

Если мы добавим код

```
var x=anEl.attributes.removeNamedItem("position");
alert(anEl.attributes.item(1));
```

функция `alert` покажет `null`, поскольку атрибут был удален.

Однако если мы вместо этих строк кода добавим другой код,

```
var x=anEl.attributes.removeNamedItem("type");
alert(anEl.attributes.item(1));
```

функция `alert` возвратит значение `first`, указывая тем самым, что порядок расположения атрибутов в объекте `NamedNodeMap` не был изменен. Это демонстрирует разницу между объектами `NamedNodeMap` и `NodeList`, где порядок расположения изменен.

Методы сборки интерфейса Node

Методы сборки интерфейса `Node` (табл. 6.5) позволяют вставлять узлы, которые уже были созданы с помощью методов сборки интерфейса `Document` (описанных ниже в разделе «Интерфейс Document»), в соответствующее место документа.

В следующем разделе «Интерфейс Document» мы приведем пример, который поможет вам понять, как работают эти методы.

Теперь рассмотрим интерфейс `Document`, предоставляемый объектной моделью документа.

Интерфейс Document

Доступные только для чтения атрибуты интерфейса Document

Ниже (табл. 6.6) приведен список доступных только для чтения атрибутов интерфейса `Document`.

Таблица 6.5. Методы сборки интерфейса Node

Метод	Параметры	Возвращаемые значения	Комментарии
insertBefore (вставить перед)	Имеет два параметра: 1. Узел, который нужно вставить 2. Узел, перед которым его нужно вставить	Вставляемый узел	
replaceChild (заменить дочерний узел)	Имеет два параметра: 1. Узел, который нужно вставить в список дочерних узлов 2. Узел, который нужно заменить	Замещенный узел	
removeChild (удалить дочерний узел)	Узел, который нужно удалить	Удаленный узел	
appendChild (добавить дочерний узел)	Узел, который нужно добавить	Добавленный узел	Добавляет узел в конец списка узлов
hasChildNodes (имеет ли IE5. дочерние узлы) метод логического типа	Не имеет параметров	Возвращает значение true, если узел имеет дочерние узлы, в противном случае возвращает значение false	Этот метод не поддерживается в ранних бета-версиях браузера Вместо применения этого метода следует проверять значение атрибута length дочернего узла
cloneNode (узел-двойник)	deep=(true|false)	узел-двойник	Если параметр deep (глубина)узла-двойника принимает значение true, то все дочерние узлы данного узла дублируются. В противном случае дублируется только сам узел с его атрибутами (если он является элементом)

Таблица 6.6. Атрибуты интерфейса Document, доступные только для чтения

Атрибут	Тип	Комментарии
doctype типа документа (тип документа)	DocumentType	Этот атрибут обеспечивает доступ к определению
implementation (реализация)	DOMImplementation	Этот атрибут имеет метод hasFeature(), который возвращает логическое значение. Этому методу передается два аргумента: аргумент feature (свойство) и аргумент version (версия).
documentElement (элемент документа)	Element	Этот атрибут возвращает корневой элемент документа

Атрибут doctype

Этот атрибут обеспечивает доступ к внутреннему определению типа документа. В результате добавлени в раздел `SCRIPT` файла `DOMmain.htm` кода

```
alert(MyDoc.doctype);
```

на экране появится сообщение, показанное на рис. 6.12.

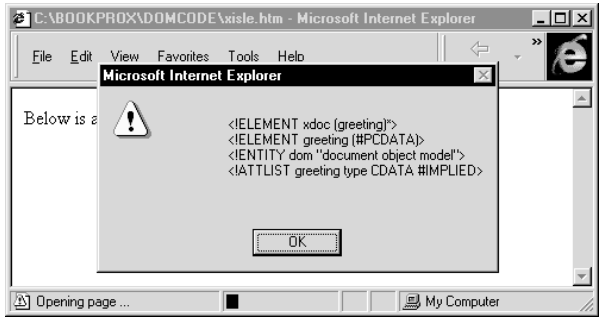


Рис. 6.12. Действие атрибута doctype в файле с встроенным определением (DTD)

Если мы перепишем участок, написанный на XML, в файле `DOMmain.htm` следующим образом:

```
<XML ID="xisle">
       <?xml version="1.0"?>
       <!DOCTYPE xdoc SYSTEM "xisle.dtd"
       [
       <!ENTITY dom "объектная модель документа">
       ]>
       <!-- Первый комментарий. -->
       <xdoc>
       <greeting type="cordial" position="first">&amp; Здравствуй, &dom;</
greeting>
       <greeting> Здравствуй, XML</greeting>
       <!-- Комментарий. -->
       <?pi сделай это и вот это?>
   </xdoc>
</XML>
```

и в той же директории поместим отдельный файл `xisle.dtd` следующего содержания:

```
<!ELEMENT xdoc (greeting)+>
<!ELEMENT greeting (#PCDATA)>
<!ENTITY amp "&#38;#38;">
<!ATTLIST greeting
    type CDATA #IMPLIED
>
```

то получим эквивалентный документ. Однако теперь тот же самый код

```
alert(MyDoc.doctype);
```

приведет к другому результату (рис. 6.13).

Другими словами, все, что хранит атрибут `doctype` – это внутреннее подмножество DTD. Смысл его будет более понятен после рассмотрения таких понятий как *компоненты*.

Существуют случаи, когда бывает необходимо получить доступ ко всему определению типа документа, например, когда по ходу дела требуется изменить содежание какого-нибудь элемента. Первый уровень DOM не предоставляет соответствующих интерфейсов для такого случая.

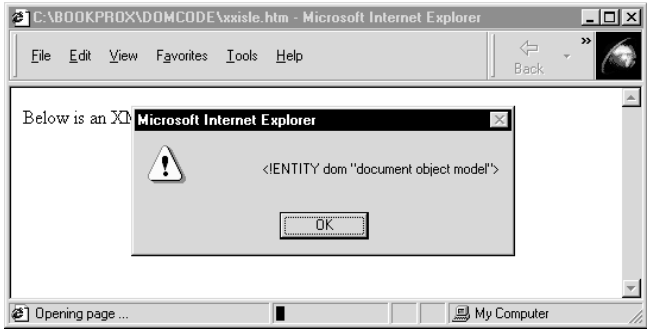


Рис. 6.13. Действие атрибута doctype в файле с внешним определением (DTD)

Атрибут Implementation

Этот атрибут возвращает интерфейс `DOMImplementation`, характеризующий тип реализации документа. Интерфейс `DOMImplementation` предоставляет метод `hasFeatures()`, который возвращает логическое значение. Метод имеет два аргумента: `feature` и `version`.

Аргумент `feature` может принимать значение `'XML'` или `'HTML'` (регистр не имеет значения). В случае XML номер версии должен быть таким же, какой указан в объявлении версии.

Помещенный в разделе `SCRIPT` файла `DOMmain.htm` код

```
MyDoc.implementation.hasFeature("XML","1.0");
```

возвращает значение `true`, а любой другой, включая

```
MyDoc.implementation.hasFeature("XML","1");
```

возвращает значение `false`, поскольку тестируется значение символьной строки, а не число.

Как уже говорилось, проверка версии имеет значение при управлении документами, и несмотря на то, что в случае простой HTML-страницы нам, возможно, эта операция и не потребуется, – в других случаях без нее не обойтись. Например, такая необходимость возникает, когда требуется рассортировать весь банк документов на документы XML и HTML.

Атрибут documentElement

Этот атрибут возвращает корневой элемент документа. Очевидно, что доступ к нему очень важен, поскольку именно в корневом элементе практически размещается содержание документа. Таким образом, с его помощью можно получить доступ ко всему остальному документу.

Добавление следующего кода в раздел `SCRIPT` файла `DOMmain.htm`

```
var rootEl=myDoc.documentElement;
```

приведет к созданию переменной `rootEl`, содержащей все узлы, находящиеся после пролога XML. Затем можно воспользоваться атрибутом `childNodes` нашего интерфейса (напоминаем, что интерфейс `Document` наследует этот атрибут из основного интерфейса `Node`) для получения доступа ко всем остальным узлам документа.

Методы сборки в интерфейсе Document

Несмотря на то, что во время создания некий узел не имеет родительских узлов и узлов-братьев, все равно фактически он принадлежит документу. Чтобы внедрить созданный узел в структуру документа, необходимо применить унаследованные от интерфейса `Node` методы сборки. Однако необходимо учитывать, что уже при создании узел является объектом со своими собственными правами.

Все методы интерфейса `Document` (табл. 6.7) являются методами сборки, мы используем их для создания каждого типа узла.

Таблица 6.7. Методы интерфейса Document

Метод	Допустимые параметры	Тип возвращаемого узла	Комментарии
createElement (создать элемент)	Строка, представляющая собой имя тэга элемента	Element	
createDocument Fragment (создать фрагмент него документа)	Нет	DocumentFragment	Просто создает фрагмент документа. Элементы могут быть добавлены в с помощью методов интерфейса Node
createTextNode (создать текстовый узел)	Строка	Text	Строка является значением текстового узла, или, другими словами, тем, что должно быть в нем написано
createComment (создать комментарий)	Строка	Comment	Знаки <!-- и --!> добавляются автоматически. Параметром должен быть текст между ограничителями
createCDATA Section (создать секцию CDATA)	Строка	CDATASection	
createProcessing Instruction (создать команду приложения)	Две строки, одна из которых адресат команды, вторая – данные для узла	Processing Instruction	Знаки <? и ?> добавляются автоматически, так же как и для комментариев
createAttribute (создать атрибут)	Строка	Attr	Возможно, лучше создавать атрибуты и их значения с помощью метода setattribute интерфейса Element (рассмотрен далее в разделе «Интерфейс Element»)
createEntity Reference (создать ссылку на компонент)	Строка с названием компонента	EntityReference	Знаки & и ; добавляются автоматически
GetElements ByTagName (получить элемент по имени его тэга)	Имя тэга	NodeList (Список узлов) всех элементов с данным именем	Доступ к этому списку узлов можно получить с помощью индекса

Рассмотрим примеры применения этих методов.

Методы интерфейсов Node и Document

В следующем примере приведен код, который нужно поместить в файл `DOMmain.htm` для управления объектной моделью документа. Обратите внимание, что единственной целью этого примера является демонстрация синтаксиса.

Вначале мы создадим три новых узла: `greeting`, `other` и узел комментария, затем:

1. Добавим один из узлов – узел `comment`.
2. Создадим дубликат узла `comment` и добавим этот дубликат.
3. Вставим элемент `other` в начало документа.
4. Заменим узел команды приложения узлом `greeting`.

Скрипт, приведенный ниже, содержит комментарии к происходящим действиям.

```
<SCRIPT>
// Создает объект XML-документа.
   myDoc=xisle;    // xisle - это участок XML в исходном HTML
                   // документе, приведенном в разделе 'Интерфейсы
                   // Node и Document'.
   rootEl=myDoc.documentElement;
   anEl=rootEl.firstChild;
// Методы сборки document и node.
   document.write("<H1 align='center'>Демонстрация методов сборки</H1>");
// Создаем новые элементы.
   x=myDoc.createElement("greeting");
   y=myDoc.createElement("other");
   z=myDoc.createComment("A created comment");
// Создаем список узлов.
   nlist=rootEl.childnodes;
// Записываем первоначальный список узлов.
   writechildnodes(nlist);
// Добавляем новые узлы.
   dummy=rootEl.appendChild(z);
// Записываем новый список узлов.
   writechildnodes(nlist);
// Дублируем узел и вставляем его в новые узлы.
   var zz=z.cloneNode(false);
   dummy=rootEl.appendChild(zz);
// Записываем новый список узлов.
   writechildnodes(nlist);
// Вставляем элемент 'other'.
   dummy=rootEl.insertBefore(y,anEl);
// Записываем новый список узлов.
   writechildnodes(nlist);
// Заменяем pi на x.
   dummy=rootEl.replaceChild(x,nlist(4));
// Записываем новый список узлов.
```

```
    writechildnodes(nlist);
    //----- Эта функция записывает узлы. ------
    function writechildnodes(nlist2)
    {
        len=nlist2.length;
        document.write("<BR>");
        for(var i=0;i<len;i++)
        {
        document.write(nlist(i).noeName + " | ");
        }
    }
    //----- Конец функции. ------
</SCRIPT>
```

На экране должно появиться изображение, приведенное на рис. 6.14.

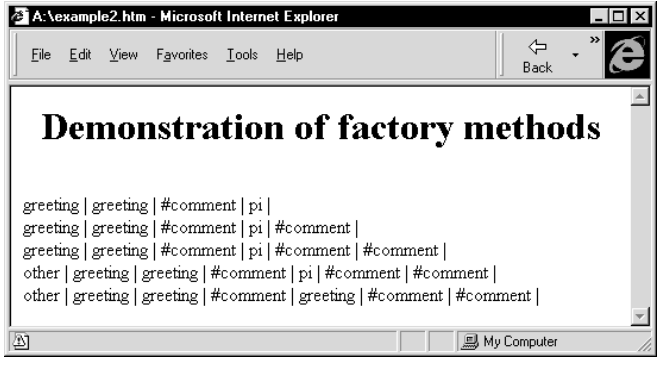


Рис. 6.14. Демонстрация действия методов интерфейса Node и Document

Обратите внимание:

- как мы применяем простую функцию `writechildnodes`, которая использует оператор `for` языка JavaScript, для того чтобы продемонстрировать имена узлов на экране;
- как список узлов `nlist` автоматически обновляется всякий раз, как в него добавляется узел; нет необходимости писать код его обновления;
- как и в методе `cloneNode()`, используется параметр `true` или `false`. Это определяет «глубину» дублирования: `deep='true'` или `deep='false'`. В этом примере мы продублировали комментарий, который не может иметь дочерних узлов. Но если бы это был элемент, то с помощью значения `deep='true'` мы бы продублировали все его дочерние объекты. В случае `deep='false'` были бы продублированы только его атрибуты и их значения.

Мы рассмотрели примеры только некоторых методов и атрибутов интерфейсов `Document` и `Node`. Надеемся, что у вас не будет трудностей с использованием и других их свойств.

Интерфейс CharacterData

Интерфейс `CharacterData` предоставляет целую серию методов и два атрибута для управления строками в любых узлах, имеющих символьные данные, например в текстовом узле и узле комментария. В табл. 6.8 приведены возможные методы и атрибуты. Атрибуты помечены.

Таблица 6.8. Методы и атрибуты интерфейса CharacterData

Методы или атрибуты	Допустимые параметры	Возвращаемое значение	Комментарии
data (данные) (атрибут)	Нет	Строка – содержание узла	В случае применения к команде приложения возвращает все, что находится между <? и ?>, а при применении к комментарию возвращает все, что находится между <!-- и --!>. При применении к текстовому узлу возвращает текст
length (длина) (атрибут)	Нет	Число символов в строке узла	
substring Data (данные подстроки) ком	Два целых значения: 1 – позиция начала подстроки, которую	Строка, начинающаяся с указанной позиции, с длиной, равной числу,	Если число символов, которые нужно извлечь, превышает длину строки, возвращается строка цели-
	нужно извлечь; 2 – число символов, которое нужно извлечь	указанному во втором параметре	
appendData (добавить данные)	Строка, которую	Ничего не возвращает нужно добавить	
	в конец узла		
insertData (вставить данные)	Два параметра:	Ничего не возвращает позиция, с которой	Исключительная ситуация возникает, если сдвиг отрица-
тельный			
	начинается вставка строки, и строка, которую нужно вставить		или больше, чем длина узла
DeleteData (удалить данные)	Два параметра: позиция,	Ничего не возвращает	Исключительная ситуация возникает, если сдвиг отрицатель-
ный			
	с которой начинается удаление строки, и число символов, которые нужно удалить		или больше, чем длина узла, а также если количество удаляемых символов отрицательно
ReplaceData (заменить данные)	Три параметра: позиция, с которой начинается замещение строки, число удаляемых символов и строка, которой следует заменить удаленные символы	Ничего не возвращает	Если число удаляемых символов превышает расстояние до конца строки, то заменяются все символы, начиная с указанного сдвига

Попробуем снова проиллюстрировать этот тезис с помощью нескольких примеров. Вставим следующий скрипт в файл `DOMmain.htm`, приведенный выше в разделе «Интерфейсы Document и Node». На этот раз воспользуемся нашим вторым элементом `greeting` и создадим объект из его дочернего узла `text`.

Затем:

1. Измерим его длину.
2. Распечатаем его данные.
3. Извлечем из него подстроку.
4. Добавим к нему данные.
5. Вставим в объект некоторый текст.
6. Заменим часть текста.
7. Удалим часть текста.

Снова подчеркнем, что единственной целью этого примера является демонстрация синтаксиса DOM.

```
<SCRIPT>
  // Создает объект XML-документа.
  myDoc=xisle
  rootEl=myDoc.documentElement;
  // Создает объект текстового узла 'Здравствуй, XML'.
  anEl=rootEl.firstChild.nextSibling.firstchild;

  // characterData
  document.write("<H1 ALIGN ='center'>Examples CharacterData Interface</H1>")
  document.write("<BR><SPAN STYLE='font-size:14pt;'>Example of length :- </SPAN>")
  document.write(anEl.length)
  document.write("<BR><SPAN STYLE='font-size:14pt;'>Example of data :- </SPAN>")
  document.write(anEl.Data)
  document.write("<BR><SPAN STYLE='font-size:14pt;'>Example of a substring :- </SPAN>")
  document.write(anEl.substringData(6,3))
  document.write("<BR><SPAN STYLE='font-size:14pt;'>Example of appendData :- </SPAN>")
  dummy=anEl.appendData(" More Text!!");
  document.write(anEl.data);
  document.write("<BR><SPAN STYLE='font-size:14pt;'>Example of insertData :- </
SPAN>")
  dummy=anEl.insertData(5," Inserted Text!! ")
  document.write(anEl.data);
  document.write("<BR><SPAN STYLE='font-size:14pt;'>Example of replaceData :- </SPAN>")
  dummy=anEl.replaceData(21,4,"Core1 level XML DOM!!")
  document.write(anEl.data);
  document.write("<BR><SPAN STYLE='font-size:14pt;'>Example of deleteData :- </SPAN>")
  dummy=anEl.deleteData(21,11)
  document.write(anEl.ata);
</SCRIPT>
```

В этом коде, кажется, нет никаких «подводных камней», все очевидно.

Результат действия кода продемонстрирован на рис. 6.15.

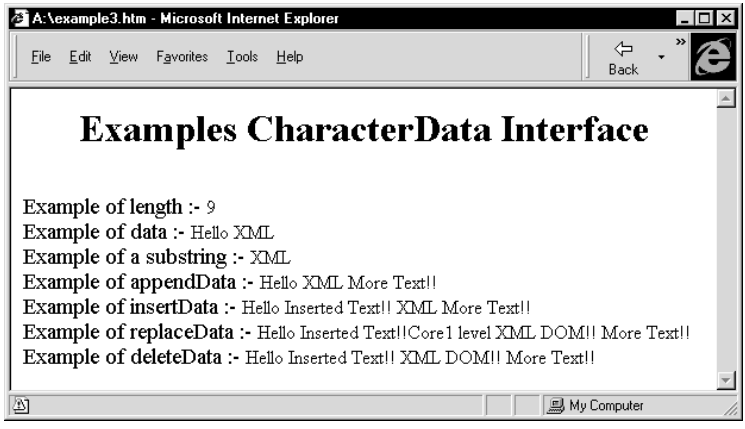


Рис. 6.15. Демонстрация действия методов интерфейса CharacterData

Теперь перейдем к интерфейсу `Attr`.

Интерфейс Attr

Атрибуты элемента в объектной модели документа, разработанной консорциумом W3C, не являются частью дерева документа, а скорее рассматриваются как часть элемента.

Уточнение *Интерфейс Attr (который представляет атрибут XML) в ранних версиях DOM назывался интерфейсом attribute. Его название было изменено на Attr, чтобы избежать путаницы с ключевым словом языка IDL attribute, которое используется для обозначения полей интерфейсов.*

Поскольку атрибуты являются частью элемента, такие методы как `parentNode` и `nextSibling` будут возвращать значение `null` при их использовании с этим интерфейсом.

Атрибуты интерфейса `Attr` приведены в табл. 6.9.

Таблица 6.9. Атрибуты интерфейса Attr

Атрибуты	Возвращаемые значения	Комментарии
name (имя)	Имя атрибута	
specified (как задается)	true или false	Возвращает значение true, если в первоначальном документе значение атрибута задано явно. Возвращает false, если атрибут принимает значение по умолчанию
value (значение)	Значение атрибута	

Если в нашем файле `DOMmain.htm` добавить в раздел `SCRIPT` код:

```
myDoc=xisle;
rootEl=myDoc.documentElement;
```

```
anEl=rootEl.firstChild;
x=anEl.attributes(0);
document.write(x.specified+ "<BR>");
document.write(x.nodeValue +"<BR>");
document.write(x.nodeName + "<BR>");
alert(x.parentNode);
alert(x.nextSibling);
```

то обе функции `alert` возвратят `null`, а браузер покажет:

```
true
cordial
type
```

Теперь перейдем к интерфейсу `Element`.

Интерфейс Element

Большинство узлов документа является текстовыми узлами и элементами. Поскольку методы интерфейса `Node` могут быть использованы для всех элементов, единственные методы, которые связаны напрямую с интерфейсом `Element`, – это методы, касающиеся атрибутов и их значений.

Мы уже видели, как код со следующим синтаксисом:

```
X=[nodeobject].attributes;
```

формирует объект `NameNodeObject X`, а код с синтаксисом:

```
[document object].createAttribute("[имя тэга]");
```

создает одиночный узел `Attr`, который может быть присвоен элементу с помощью одного из методов, о которых мы сейчас и поговорим.

Методы и атрибуты интерфейса `Element` приведены в табл. 6.10.

Вот несколько листингов, поясняющих приведенные методы. Подчеркнем еще раз – единственной целью этого примера является демонстрация синтаксиса DOM. Вставьте в файл `DOMmain.htm` следующий участок кода:

```
<SCRIPT>
   // Создает объект XML-документа.
   myDoc=xisle;

   rootEl=myDoc.documentElement;
   // Создает объект из первого элемента greeting.
   anEl=rootEl.firstChild;
   document.write("<DIV STYLE='font-size:22pt;'>")
document.write("<SPAN STYLE='font-size:14pt;'>We use getAttribute to get the first
'greeting' type atribute value </SPAN><BR>")
. . .
```

Следующий участок кода просто получает значение атрибута и записывает его:

```
. . .
   document.write(anEl.getAttribute("type")+"<BR");
. . .
```

Таблица 6.10. Методы и атрибуты интерфейса Element

Методы или атрибуты	Допустимые параметры	Возвращаемое значение	Комментарии
tagName (имя тэга)	Нет	Имя тэга элемента	Возвращает такое же значение, как если бы был использован метод интерфейса Node [nodeobject].nodeName
getAttribute (получить атрибут)	Имя атрибута	Значение атрибута	Заметим, что это единственный способ в текущей DOM получить значение атрибута, если только не выделять узел Attr (смотрите метод GetAttributeNode ниже) или не использовать NameNodeList
setAttribute (установить атрибут)	Два параметра: 1 – имя атрибута, 2 – строка, которая будет значением атрибута.	Нет	Если атрибут уже существует, этот метод просто заменит его значение. Однако если атрибута с таким именем нет, то этот метод создаст атрибут с таким именем и присвоит ему указанное значение. Если имя атрибута не разрешено в DTD, возникнет ошибка
removeAttribute (удалить атрибут)	Имя атрибута, который нужно удалить	Нет	
getAttributeNode (получить узел-атрибут)	Имя атрибута, который нужно восстановить	Имя узла Attr	К возвращаемому объекту могут быть применены методы nodeType, nodeName, nodeValue
setAttributeNode (установить узел-атрибут)	Имя узла атрибута, который должен быть добавлен в список атрибутов	Если атрибут замещает атрибут с таким же именем, то возвращается старый атрибут, в противном случае возвращается null	Если имя атрибута не разрешено в DTD, возникнет ошибка
removeAttributeNode (удалить узел-атрибут)	Узел Attr, который должен быть удален из списка	Удаляемый узел	
getElementsByTagName (получить элементы по имени тэга)	Имя соответствующего	Список всех элементов (Nodelist) с данным тэга	Доступ к каждому отдельному узлу может быть получен именем с помощью индекса
normalize (нормализовать)	Нет	Нет	Этот метод объединяет все соседние дочерние текстовые узлы в элемент. Смотрите примечание, приведенное ниже

Приведенный далее участок кода использует метод `setAttribute` для изменения значения атрибута `type` и записывает новое значение.

```
. . .
   dummy=anEl.setAttribute("type","cold");
document.write("<SPAN STYLE='font-size:14pt;'>We have used setAttribute
to change cordial to cold </SPAN><BR>")
   document.write(anEl.getAttribute("type")+"<BR>");
. . .
```

Далее создаем новый атрибут с именем `class` и значением `simple`

```
. . .
   dummy=anEl.setAttribute("class","simple");
document.write("<SPAN STYLE='font-size:14pt;'>We use setAttribute to make
a new attribute with a value of 'simple'</SPAN><BR>")
   document.write(anEl.getAttribute("class")+"<BR>");
. . .
```

Ниже формируем новый элемент, используя метод `createElement` интерфейса `Document`. Затем добавляем этот элемент в конец дочерних элементов узла `xdoc`:

```
. . .
   x=myDoc.createElement("other");
   dummy=rootEl.appendChild(x)
document.write("<SPAN STYLE='font-size:14pt;'>We have created an element called
other </SPAN><BR>")
   document.write(rootEl.lastChild.nodeName+"<BR>")
. . .
```

Теперь создадим атрибут `junk`, используя метод `createAttribute` интерфейса `Document`. Затем применим метод `setAttributeNode`, чтобы добавить этот атрибут элементу `other`, который мы ранее прочитали, в переменную `x`. После этого используем метод `getAttribute`, чтобы продемонстрировать отсутствие значения у атрибута `junk`. Затем применим метод `setAttribute` для присвоения этому атрибуту значения, и снова применим метод `getAttribute`, чтобы убедиться, что присваивание выполнено.

```
. . .
   y=myDoc.createAttribute("junk");
   dummy=x.setAttributeNode(y)

document.write("<SPAN STYLE='font-size:14pt;'>We are getting the value of the
'junk', we created and set it into other element. (Shold be null) </SPAN><BR>")
   document.write(x.getAttribute("junk")+"<BR>");
   dummy=x.setAttribute("junk","junkvalue");
document.write("<SPAN STYLE='font-size:16pt;'>We gave it a value of 'jnkvalue' </
SPAN><BR>")
   document.write(x.getAttribute("junk")+"<BR>");
   document.write("</DIV>")
</SCRIPT>
. . .
```

Вот как это выглядит на экране (рис. 6.16).

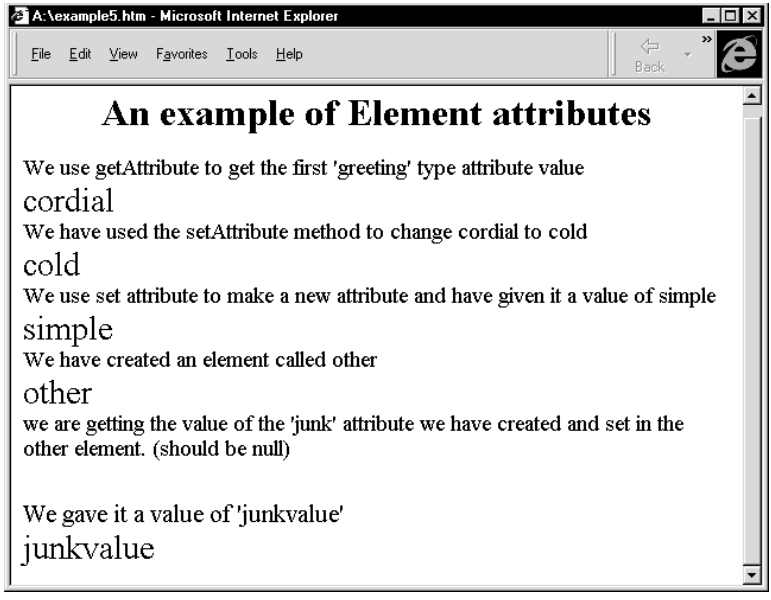


Рис. 6.16. Демонстрация действия атрибутов интерфейса Element

Далее рассмотрим два оставшихся метода интерфейса `Element`.

Метод getElementsByTagName

Этот метод создает список узлов всех элементов с данным именем. Доступ к каждому отдельному узлу может быть получен с помощью индекса.

Код:

```
var greetlist=myDoc.getElementsByTagName("greeting");
```

приведет к созданию индексированного списка узлов `Nodelist` всех элементов `greeting`.

Метод normalize

В процессе создания документа с помощью методов сборки может оказаться, что строка текста построена несколькими повторно примененными методами `createTextNode` и `appendChild`.

Поэтому вполне возможно, что элемент `<macbeth>`

```
<macbeth> "Tomorrow, and tomorrow, and tomorrow creeps on our petty pace from day
to day"</macbeth>
```

состоит не из одного узла, а из нескольких!

```
<macbeth> "tomorrow, | and tomorrow, | and tomorrow | creeps on our petty pace |
from day to day"</macbeth>
```

Это может привести к путанице при использовании, например, указателей XPointer. Чтобы объединить все эти дочерние узлы в один дочерний узел элемента `macbeth`, напишем еще две строки (предполагая, что это четыреста тридцать второй элемент `macbeth`).

```
maclist=shakedoc.getElementsByTagName('macbeth');
macbeth(432).normalize;
```

Теперь вместо пяти дочерних узлов мы имеем один! Таким образом, нам удалось свернуть соседние текстовые узлы в один-единственный узел.

Следует заметить, что если документ, созданный методом сборки, периодически сохраняется, такие дочерние узлы будут объединяться, поскольку сохранение должно приводить к вынужденной нормализации.

Теперь переходим к интерфейсу текстового узла.

Интерфейс узла Text

Интерфейс узла `Text` наследует все методы и атрибуты, имеющиеся у интерфейса `CharacterData`. Это означает, что все методы, которые мы рассмотрели в рамках интерфейса `CharacterData`, могут быть использованы для управления любыми текстовыми узлами.

Вдобавок, интерфейс `Text` имеет собственный метод, называемый `splitText` (расщепление текста).

Метод splitText

Этот метод делит один текстовый узел на два отдельных. В нем в виде целого числа содержится единственный параметр, который является величиной сдвига от начала текстовой строки. Таким образом, если объект `txtObj` представляет собой следующую строку:

```
"Время рождаться и время умирать"
```

то код:

```
newTxtObj=txtObj.splitText(15);
```

приведет к созданию двух текстовых объектов: `newTxtObj` (его значением будет `"и время умирать"`), и `txtObj` (`"Время рождаться"`).

Мне трудно представить, при каких обстоятельствах этот метод может быть использован, однако он существует.

Интерфейс Comment

Интерфейс `Comment` наследует все методы интерфейса `CharacterData`, но не имеет своих собственных.

Все методы, которые мы рассматривали в рамках интерфейса `CharacterData`, могут быть использованы для обработки любой текстовой строки комментария, находящейся между ограничителями `<!--` и `--!>`.

Интерфейс Processing Instruction

Команда приложения (Processing Instruction, PI) представляет собой способ обработки специальной информации в тексте документа. Она состоит из адресата и данных. Команда приложения, которая обычно встречается чаще других, связывает документ и таблицу стилей.

```
<?xml:stylesheet type="text/css" href="myxml_style.css"?>
```

В этой команде приложения адресатом является `xml:stylesheet`, а все, что находится после первого пробела, преставляет собой данные.

Интерфейс PI имеет два доступных только для чтения атрибута: `target` и `data`. Ими можно управлять с помощью методов интерфейса `CharacterData`.

Предполагая, что в нашем XML-документе имеется команда приложения для таблицы стилей, напишем следующий код

```
var targetStr=myPI.target;
```

Он приведет к созданию текстового объекта `targetStr`, где содержанием будет `'xml:styleheet'`. А включив строку:

```
var dataStr=myPI.data;
```

в результате получим текстовой объект `dataStr`, в котором будет содержаться `'href="myxml_style.css" type="text/css" '`.

Интерфейс DocumentType

Мы уже видели, как при использовании интерфейса `Document` атрибут

```
X=myDoc.doctype;
```

присваивает объект `DocumentType` переменной `X`.

Интерфейс `DocumentType` содержит доступные только для чтения атрибуты, перечисленные в табл. 6.11.

Таблица 6.11. Атрибуты интерфейса DocumentType

Атрибуты	Возвращаемые значения	Комментарии
name (имя)	Имя документа	Имя является строкой, непосредственно следующей за декларацией <!DOCTYPE, то есть за строкой xdoc в нашем примере. Заметим, что то же самое значение используется при применении атрибута nodeName интерфейса Node
entities (компоненты)	NameNodeMap – карта всех компонентов	Доступ к этой карте можно получить с помощью индекса, несмотря на то, что порядок следования не имеет значения. Смотрите описанный ранее интерфейс NameNodeMap
notations (обозначения)	NameNodeMap – карта всех обозначений	

Добавим следующий участок кода в наше определение типа документа для островка XML в файле `DOMmain.htm`:

```
<!ATTLIST greeting
   type CDATA #IMPLIED
   class CDATA #IMPLIED
   position CDATA #IMPLIED
>
<!ENTITY dom "document object model">
<!ENTITY htm SYSTEM "core2.xml">
<!NOTATION gif SYSTEM "http://someserver.com/gwswin/gws.exe">
```

и посмотрим, можно ли использовать интерфейсы DocumentType, Entities и Notation для доступа к информации.

В нашем примере следующий код в разделе SCRIPT файла DOMmain.htm приведет к созданию карты NamedNodeMap всех компонентов:

```
document.write(docObj.entities(0).nodeName);
document.write(docObj.entities(1).nodeName);
```

и напечатает имена dom и htm соответственно.

Код:

```
document.write(docObj.notations(0).nodeName);
```

напечатает имя gif.

Интерфейс Notation

Интерфейс Notation имеет два доступных только для чтения атрибута: publicId и systemId.

Если его применить совместно с кодом предыдущего раздела:

```
document.write(docObj.notations(0).systemId);
```

это приведет к выводу на экран адреса http://someserver.com/gwswin/gws.exe.

Интерфейс Entity

Интерфейс Entity имеет три доступных только для чтения атрибута: publicId, systemId и notationName.

При использовании совместно с измененным определением типа документа файла DOMmain.htm, приведенным ранее в разделе «Интерфейс DocumentType», благодаря строке:

```
document.write(docObj.entities(1).systemId);
```

будет распечатано значение core2.xml.

Атрибут notationName возвращает имя обозначения необрабатываемого компонента. Поскольку оба компонента обрабатываются, возвращается null.

Интерфейс EntityReference

Способ, с помощью которого агент пользователя обрабатывает встречающиеся ему ссылки на компоненты, во многом зависит от самого агента.

Агент пользователя должен раскрывать все предопределенные компоненты, такие как амперсанд, одиночные и двойные кавычки, символы «больше» и «меньше», а также, в большинстве случаев, все символьные компоненты.

На рис. 6.17 показано, как браузер IE5 раскрывает следующий скрипт:

```
for(i=930; I<976; I++)
   {
   dcument.write("&#"+ i +";");
   }
```

Раскрытие компонента, если оно возможно, проявляется в виде дочернего узла ссылки на компонент. Поэтому в файле DOMmain.htm код

```
x=anEl.lastChild;
```

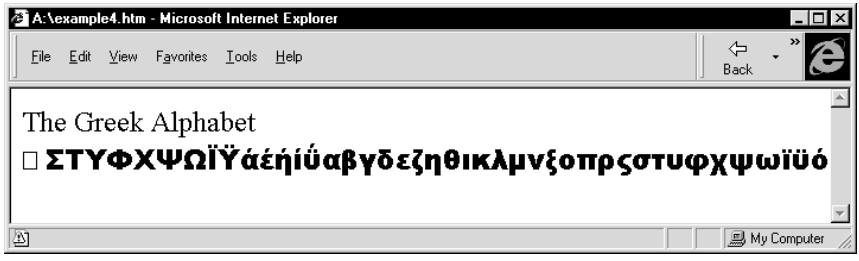


Рис. 6.17. Демонстрация интерфейса EntityReference. Вывод греческого алфавита

создает объект ссылки на компонент `&dom;`, а функция

```
alert(x.firstChild);
```

демонстрирует раскрытие компонента `"document object model"` (объектная модель документа).

Заключение

На этом завершается обзор объектной модели документа, утвержденной консорциумом W3C, и ее реализации в браузере IE5. Объектная модель документа предоставляет независимый от платформы и приложения способ обработки XML-документов.

Надеемся, что скоро появятся приложения, которые позволят сохранять обработанный документ. Несмотря на то, что современные браузеры не в состоянии поддерживать эти операции, их, тем не менее, можно использовать для проведения довольно сложных манипуляций с XML и DOM. Далее мы рассмотрим несколько простых примеров использования объектной модели документа в браузере IE5.

Все приведенные выше примеры в практическом смысле бесполезны, они годны только для демонстрации синтаксиса DOM. Теперь обратимся к некоторым реализациям DOM, которые вполне могут пригодиться в реальной жизни.

Некоторые простые реализации

Действующая объектная модель открывает широкие перспективы в области обработки и просмотра документов. Три простых примера, приведенных ниже, демонстрируют те возможности, которых уже сейчас нетрудно добиться при помощи браузера, совместимого с XML и объектной моделью документ.

Основной рекурсивный цикл

В большинстве случаев основой обработки дерева XML является простой рекурсивный цикл, обходящий все узлы по порядку. Такой цикл показан ниже.

Функция `getchildren` кода принимает в качестве параметра узел, создает список всех дочерних узлов, проходит по этому списку, проверяя, имеет ли каждый дочерний узел свои собственные дочерние узлы, и если имеет, то вызывает сама себя. Файл `nest1.xml`, который она использует в качестве островка XML, приводится после кода HTML.

```
<HTML>
<!-- Эта простая программа загружает XML-документ в браузер IE5 и проходит все
узлы, используя рекурсивную функцию. -->
   <!-- Загружает правильный документ в островок XML. -->
   <XML ID="island1" SRC="nest1.xml"></XML>
   <SCRIPT>
// Создает объект документа и список узлов.

       var myDoc=island1;
       var myDocNodeList=myDoc.childNodes;
   if(myDoc.parseError.reason!="")
       {
       alert(myDoc.parseError.reason);
       }
   else
       {
       alert("состоятельный XML-документ");
       }
// Создает объект - корневой элемент...
       var rootEl=myDoc.documentElement;
// ..и проходит его содержание.
       x=getchildren(myDoc);
// ================ Начало рекурсивной функции 'getchildren'.==========
   function getchildren(node)
   {
// Создает список дочерних элементов.
   var x=node.childNodes;
   var z=x.length;
   if(z!=0)
       {
       for(var i=0;i<z;i++)
           {
// Вся обработка элементов XML находится здесь.
           document.write("Node Type=" + x(i).nodeType);
           document.write("Node Name=" + x(i).nodeName);
           document.write("Node Value=" + x(i).nodeValue + "<BR>");
           getchildren(x(i));
           }
       }

   }
// ========== Конец рекурсивной функции 'getchildren'.===================
   </SCRIPT>
</HTML>
```

В этом примере для обхода выбран приведенный ниже файл. Как видно на рис. 6.18, здесь обеспечена выдача детальной информации о каждом узле.

```
<xdoc>a
   <nest1>b<nonnest1/>
```

```
        <nest>c<nonnest2/>
            <nest3>d<nonnest3/>
            </nest3>
        </nest2>
    </nest1>
    <nonnest1/>
    <nonnest2/>
    <nonnest3/>
</xdoc>
```

Этот достаточно простой файл показывает, как работает рекурсия.

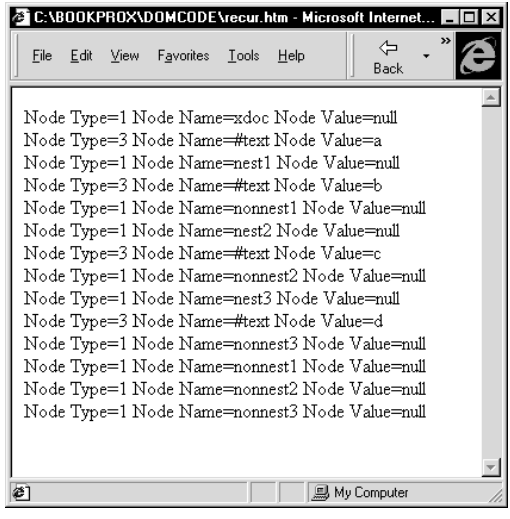


Рис. 6.18. Результат работы программы рекурсивного обхода дерева XML

Рекурсивная функция вызывает по порядку все дочерние элементы. Теперь ее или похожую функцию можно применить для реальной обработки XML-документа. Варианты этой функции будут использованы для просмотра XML-документа и применения к нему стилей, а также для демонстрации простого набора данных в виде таблицы.

Простое оформление стилями

Используя один из вариантов рекурсивной схемы, применим стиль к некоему простому документу. Для этого модифицируем рекурсивную функцию следующим образом:

```
function getchildren(node)
{
// Создаем список дочерних узлов.
var x=node.childNodes;
var z=x.length;
if(z!=0)
```

```
        {
        for(var i=0;i<z;i++)
            {
            if(x(i).nodeType==3)
                {
                document.write(x(i).nodeValue);
                getchildren(x(i));
                }
            else if(x(i).nodeType==1 && x(i).nodeName=="farewell" )
                {
                document.write(" <DIV STYLE='font-size:20pt;color:red;'> ");
                getchildren(x(i));
                document.write(" </DIV> ");
                }
         else if(x(i).nodeType==1 && x(i).nodeName=="greeting" )
                {
                document.write(" <DIV STYLE='font-size:16pt;color:green'> ");
                getchildren(x(i));
                document.write(" </DIV> ");
                }
            else
                {
                getchildren(x(i));
                }
            }
        }
    }
```

Обратите внимание на то, как:

- были записаны и открыты тэги HTML с соответствующими стилями;
- было найдено имя тэга и к нему применен соответствующий стиль;
- была вызвана функция `getchildren`, с чьей помощью можно узнать, есть ли у данного узла дочерние узлы, подлежащие обработке;
- был закрыт тэг HTML после вызова функции `getchildren`.

Применив эту функцию к простому файлу

```
<?xml version="1.0" encoding="UTF-8" ?>
<xdoc>
   <greeting>Hello DOM</greeting>
   <greeting>Hello XML</greeting>
   <farewell>Godbye HTML</farewell>
</xdoc>
```

получим на экране следующий текст, показанный на рис. 6.19.

Очевидно, что приложив еще немного усилий, можно создатьфункцию, которая считывает таблицу стилей извне. Эту таблицу стилей допускается импортировать как второй островок XML.

Выполнить указанное упражнение мы предоставляем читателю.

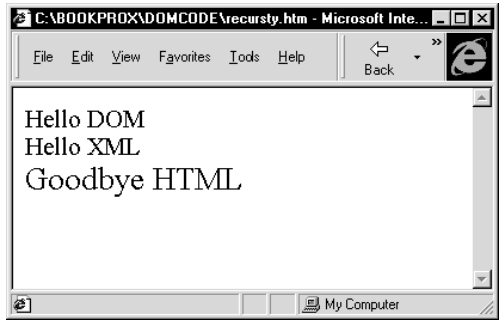


Рис. 6.19. Оформление стилями с использованием рекурсивного обхода

Простые таблицы

Данные XML-файла можно просматривать в виде простой таблицы.

Ниже приведен простой XML-файл, который можно использовать в качестве базы данных:

```
<inventory>
   <item>
       <name>Name of item</name>
       <qty>Quantity in '000's </qty>
       <cost>$'s for 10</cost>
       <sell>recommended price each item</sell>
   </item>
   <item>
       <name>Knife</name>
       <qty>100</qty>
       <cost>50</cost>
       <sell>10</sell>
   </item>
   <item>
       <name>Fork</name>
       <qty>200</qty>
       <cost>60</cost>
       <sell>10</sell>
   </item>
   <item>
       <name>Spoon</name>
       <qty>150</qty>
       <cost>70</cost>
       <sell>10 </sell>
   </item>
</inventory>
```

Далее показано, как надо модифицировать рекурсивную функцию, чтобы просмотреть этот файл.

```
function getchildren(node)
{
// Создает список дочерних узлов.
var x=node.childNodes;
var z=x.length;
if(z!=0)
    {
    for(var i=0;i<z;i++)
        {
    if(x(i).nodeType==3)
        {
        document.write(x(i).nodeValue);
        getchildren(x(i));
        }
    else if(x(i).nodeType==1 && x(i).nodeName=="inventory" )
        {
        document.write(" <TABLE BORDER=1 WIDTH=75% ALIGN='center'> ")
        getchildren(x(i));
        document.write(" </TABLE> ")
        }
    else if(x(i).nodeType==1 && x(i).nodeName=="item" )
        {
        document.write(" <TR> ")
        getchildren(x(i));
        document.write(" </TR> ")
        }
        else if(x(i).nodeType==1)
        {
        document.write(" <TD ALIGN='center'> ")
        getchildren(x(i));
        document.write(" </TD> ")
        }
        else
        {
        getchildren(x(i));
        }
        }
    }

}
```

На экран выводится таблица (рис. 6.20).

Несмотря на то, что этот скрипт создан специальнодля файла с данными указанного типа, можно, затратив еще немного усилий, написать универсальный скрипт, который использует таблицу стилей и демонстрирует любые данные в любой форме.

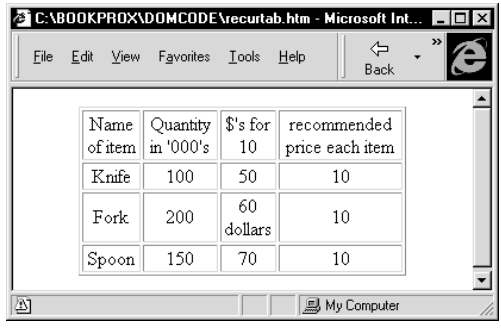


Рис. 6.20. Формирование таблицы при помощи рекурсивной функции

Подготовка слайдов

Как и у многих других лекторов, у меня скопилось больше тысячи слайдов. Как содержать их в порядке? Как облегчить поиск? Начиная с какого-то момента, я начал создавать документы XML, в которых храню тексты. Подобный подход позволил разделить слайды на категории. В первое время я пользовался простой программой, написанной на языке Visual Basic. Она подготавливала слайды для просмотра либо путем создания файлов в формате bitmap, либо путем преобразования их в формат HTML.

Скоро мне пришло в голову, что тексты слайдов можно формировать «на лету», используя объектную модель документа. Далее рассмотрены принципиальные основы нового подхода.

Шаблон для слайдов

Первое, что нужно сделать в любом новом проекте с применением XML, это понять, что именно необходимо поместить в документ. Поразмыслив, я пришел к выводу, что 99% моих слайдов состоят из следующих элементов:

- *маркированный список* (bulleted list) пунктов;
- *предложение* (statement), например, строка текста или цитата;
- *название нового раздела* (announcement), которое должно быть написано большими буквами, например, заголовок;
- кроме того, проектируя HTML-страницу, можно использовать тэг `object` для демонстрации «живого» кода и текста.

Вот как выглядит шаблон XML:

```
<?xml version="1.0" ?>
<!DOCTYPE slide SYSTEM "slide.dtd">
<slide>
<header>
<line></line>      <!-- line используется для простого форматирования. -->
</header>
<slidebody>
   <!-- Только один из следующих тэгов может быть использован в каждом слайде. -->
   <object></object>
```

```
    <image></image>
    <statement><line></line></statement>
    <announcement><line></line></announcement>

    <bullet-list>
        <bullet_title></bullet_title>
        <item>
        </item>
        <item>
        </item>
    </bullet-list>
</slidebody>
<footer>
</footer>
</slide>
```

Сохраните этот файл под именем `slide.htm` или загрузите его с сайта http://webdev.wrox.co.uk/books/1525/.

Примите к сведению, что даже самое простое XML-приложение лучше использовать вместе с определением типа документа. Создавая каждый новый документ, включающий в себя описание слайда, желательно обработать его верифицирующим анализатором и убедиться в том, что мы следуем правилам, которые задали сами.

Определение типа документа для файла `slide.xml` очень простое, но оно помогает убедиться, что слайд соответствует определенным правилам. В частности, обратите внимание на содержание элемента `slidebody`. Оно задает такие условия, чтобы мы не могли случайно добавить два различных типа слайда в один и тот же документ.

```
<?xml version="1.0" ?>
<!ELEMENT slide (header,slidebody,footer)>
<!ELEMENT header       (#PCDATA|line)*>
<!ELEMENT slidebody (object | image | statement | announcement | bullet-list) >
<!ELEMENT object       (#PCDATA)>
<!ELEMENT image        (#PCDATA)>
<!ELEMENT statement    (#PCDATA|line)*>
<!ELEMENT announcement (#PCDATA|line)*>
<!ELEMENT line         (#PCDATA)>
<!ELEMENT bullet-list  (bullet_title?,item)+>
<!ELEMENT bullet_title (#PCDATA)>
<!ELEMENT item         (#PCDATA)>
<!ELEMENT footer       (#PCDATA)>
<!ATTLIST slide type   CDATA #IMPLIED>
```

Сохраните этот файл под именем `slide.dtd`. Его тоже можно найти на Web-сайте издательства Wrox: http://webdev.wrox.co.uk/books/1525/.

Как уже было сказано, до перехода на объектную модель документа я использовал этот шаблон для создания слайда в формате bitmap с помощью простой программы на языке Visual Basic или генерировал HTML-документ с помощью другой программы.

Сейчас я в состоянии использовать объектную модель документа для создания слайдов «на лету» непосредственно из XML-документов. Код, с помощью которого производится эта операция, приведен ниже. Нетрудно заметить, что он является одним из вариантов рекурсивной функции, примененной в трех первых примерах.

В сущности, код делает следующее:

- создает объект XML `documentObject`;
- проходит с помощью цикла по всему документу, преобразовывая «на лету» XML в HTML;
- воспроизводит HTML, используя таблицу стилей.

Первая часть кода создает объект XML-документа из XML-файла, а также присоединяет к файлу таблицу стилей. Сохраните этот файл под именем `slides.htm`:

```
<HTML>
<!-- Эта простая программа загружает документ XML в браузер IE5 и проходит по всем
узлам с помощью рекурсивной функции.-->
<LINK REL=STYLESHEET TYPE="text/css" HREF="slide1.css" TITLE="Cnet">
   <!-- Загружаем правильный документ в островок XML. -->
   <XML ID="island1" SRC="sl_cnet5.xml"></XML>
   <SCRIPT>
   // Создает объект документа и список узлов.
       var myDoc=island1;
       var myDocNodeList=myDoc.childNodes;
   if(myDoc.parseError.reason!="")
       {
       alert(myDoc.parseError.reason);
       }
   else
       {
       alert("valid XML document");
       }
   // Создает объект корневого элемента.
       var rootEl=myDoc.documentElement;
. . .
```

Для изменения содержания слайда необходимо заменить значение атрибута SRC в следующей строке кода:

```
   <XML ID="island1" SRC="sl_cnet5.xml"></XML>
```

на имя любого другого XML-файла со слайдом.

Вторая часть кода проходит циклом по XML-файлу, преобразовывает тэг XML в соответствующий тэг HTML и демонстрирует его.

```
. . .
   // Проходим по содержанию документа.
       x=getchildren(myDoc);
// ================ Начало рекурсивной функции 'getchildren'.===================
   function getchildren(node)
   {
   // Создаем дочерний список
   var x=node.childNodes;
```

```
var z=x.length;
    if(z!=0)
    {
    for(var i=0;i<z;i++)
        {
        if(x(i).nodeType==3)
            {
            document.write(x(i).nodeValue);
            getchildren(x(i));
            }
        else if(x(i).nodeType==1 && x(i).nodeName=="title" )
            {
            document.write(" <TITLE> ");
            getchildren(x(i));
            document.write(" </TITLE> ");
            }
        else if(x(i).nodeType==1 && x(i).nodeName=="header" )
            {
            document.write(" <DIV CLASS='header'>");
            getchildren(x(i));
            document.write(" <HR></DIV> ");
            }
        else if(x(i).nodeType==1 && x(i).nodeName=="slidebody" )
            {
            document.write(" <DIV CLASS='container'>");
            getchildren(x(i));
            document.write(" </DIV> ");
            }
        else if(x(i).nodeType==1 && x(i).nodeName=="announcement" )
            {
            document.write(" <DIV CLASS='announcement'>");
            getchildren(x(i));
            document.write(" </DIV> ");
            }
        else if(x(i).nodeType==1 && x(i).nodeName=="statement" )
            {
            document.write(" <DIV CLASS='statement'>");
            getchildren(x(i));
            document.write(" </DIV> ");
            }
        else if(x(i).nodeType==1 && x(i).nodeName=="line" )
            {
            getchildren(x(i));
            document.write(" <BR>");
            }
        else if(x(i).nodeType==1 && x(i).nodeName=="footer" )
            {
            document.write(" <HR><DIV CLASS='footer'>");
            getchildren(x(i));
            document.write(" </DIV> ");
            }
```

```
            else if(x(i).nodeType==1 && x(i).nodeName=="footer2" )
                {
                document.write(" <HR><DIV CLASS='footer2'>");
                getchildren(x(i));
                document.write(" </DIV> ");
                }
            else if(x(i).nodeType==1 && x(i).nodeName=="footer3" )
                {
                document.write(" <HR><DIV CLASS='footer3'>");
                getchildren(x(i));
                document.write(" </DIV> ");
                }
            else if(x(i).nodeType==1 && x(i).nodeName=="object" )
                {
                document.write("<OBJECT DATA=" + x(i).getAttribute('source')+">");
                getchildren(x(i));
                document.write(" </OBJECT> ");
                }
            else if(x(i).nodeType==1 && x(i).nodeName=="image" )
                {
                document.write("<DIV ALIGN='center'><IMG SRC=" +
_x(i).getAttribute('source')+">");
                getchildren(x(i));
                document.write(" </DIV> ");
                }
            else if(x(i).nodeType==1 && x(i).nodeName=="bullet_title" )
                {
                document.write(" <DIV CLASS='bullet_title'>");
                getchildren(x(i));
                document.write(" </DIV> ");
                }
            else if(x(i).nodeType==1 && x(i).nodeName=="bullet-list" )
                {
                document.write(" <UL>");
                getchildren(x(i));
                document.write(" </UL> ");
                }
        else if(x(i).nodeType==1 && x(i).nodeName=="item" )
                {
                document.write(" <LI>");
                getchildren(x(i));
                document.write(" </LI> ");
                }
        else
                {
                getchildren(x(i));
                }
            }
        }
    }
// ========== Конец функции 'getchildren'.====================
```

```
    </SCRIPT>
</HTML>
```

Одним из важных преимуществ такого подхода является возможность изменения внешнего вида слайда простым изменением таблицы стилей. Для этого у меня есть набор подобных таблиц.

Совет *Это совсем не значит, что выступая перед публикой следует использовать материалы с различным оформлением. Золотое правило (одно из многих), которого должен придерживаться лектор, сводится к требованию единого оформления иллюстративного ряда, а это как раз то качество, которого CSS в сочетании с XML позволяет легко добиться.*

Ниже приведена таблица стилей `slide1.css`, которая использовалась для создания образов экрана, с которыми мы вскоре встретимся.

```
BODY{
        background-color:#000080;
        color:yellow;
    }
    UL{
        margin-top:.5cm;
        margin-left:2cm;
        font-size:18pt;
        height:60%;
        width:70%;
    }
    LI{
        line-height:36pt;
    }
    OBJECT{
        width:80%;
        height:80%;
        margin-left:10%;
        margin-right:10%;

    }
    .container{

        background-color:#04A090;
        border-style:solid;
        border-color:#FFFF00;
        border-width:2px;
        margin-top:.5cm;
        margin-right:15%;
        margin-bottom:.5cm;
        padding-bottom:.5cm;
        margin-left:15%;
        height:60%;
```

```
        width:70%;
    }
    .header{
        text-align:center;
        font-family:arial, sans-serif;
        font-size:36pt;
        text-decoration:underline;
    }
    .statement{
        text-align:center;
        font-family:Times new Roman,Times,Serif;
        font-size:24pt;
        padding-top:5%;
        height:60%;
    }
    .announcement{
        text-align:center;
        font-family:Times new Roman,Times,Serif;
        font-size:30pt;
        line-height:42pt;
        padding-top:5%;
        height:60%;
    }
    .footer{
        color:#AAAA00;
        text-align:right;
        padding-right:.25cm;
        padding-top:.2cm;
    }
```

Ниже (рис. 6.21) приведен один из слайдов доклада, с которым я недавно выступал. Файл со слайдом должен находиться в той же директории, что и таблица стилей, определение типа документа и файл `slides.htm`. Как видите, в этом слайде я использовал тэг `object` языка HTML для демонстрации «живого» кода:

```
<slide>
<title>XML</title>
<slidebody>
   <header>XML: A Brief Review </header>
   <main>
   <object source="sl_cnet5.txt"></object>
   </main>
   <footer>Frank Boumphry</footer>
</slidebody>
</slide>
```

Сохраните вторую версию этого кода в виде текстового файла `sl_cnet5.txt`, чтобы его можно было продемонстрировать так, как это сделано на экране монитора.

Как видите, слайд показывает свой собственный исходный код!

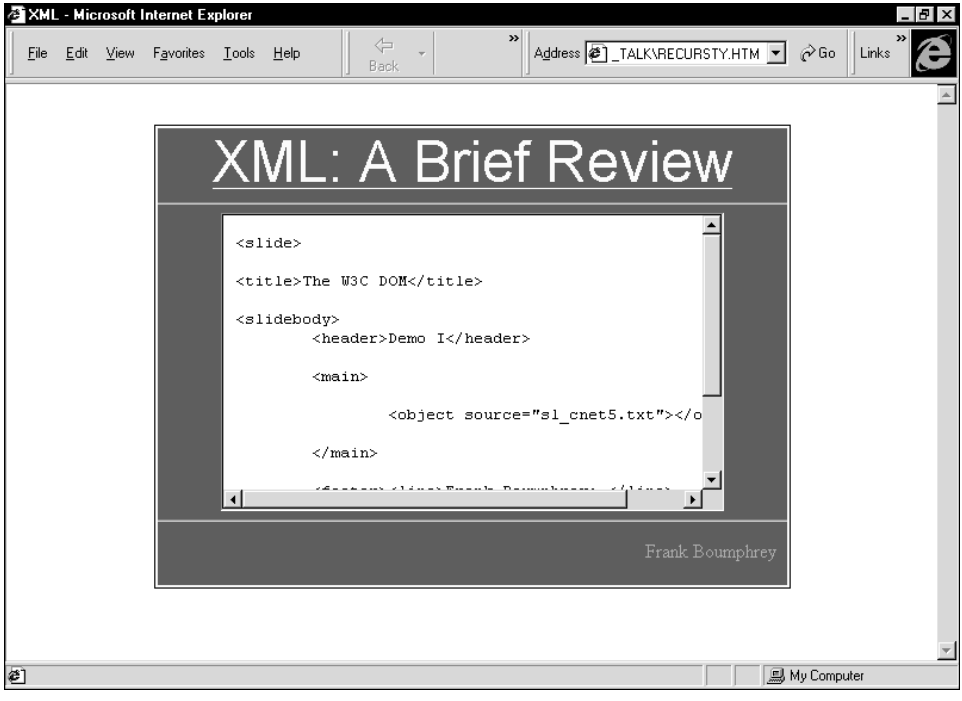


Рис. 6.21. Слайд показывает свой собственный исходный код

Ниже вы видите слайд типа «Название нового раздела» (announcement):

```
<slide>
<title>XML</title>
<slidebody>
   <header>XML: A Brief Review </header>
   <main>

   <announcement>
       <line>XML</line>
       <line></line>
       <line> The 'Well-Formed and Valid' document </line>
   </announcement>

   </main>
   <footer>Frank Boumphrey</footer>
</slidebody>
</slide>
```

Сохраните этот код в файле `sl_cnet6.xml` и измените строку кода для островка XML в файле `slides.tm` на строку:

```
<XML ID="island1" SRC="sl_cnet6.xml"></XML>
```

чтобы установить имя того слайда, который вы хотите продемонстрировать. На рис. 6.22 показано, как выглядит этот слайд на экране.

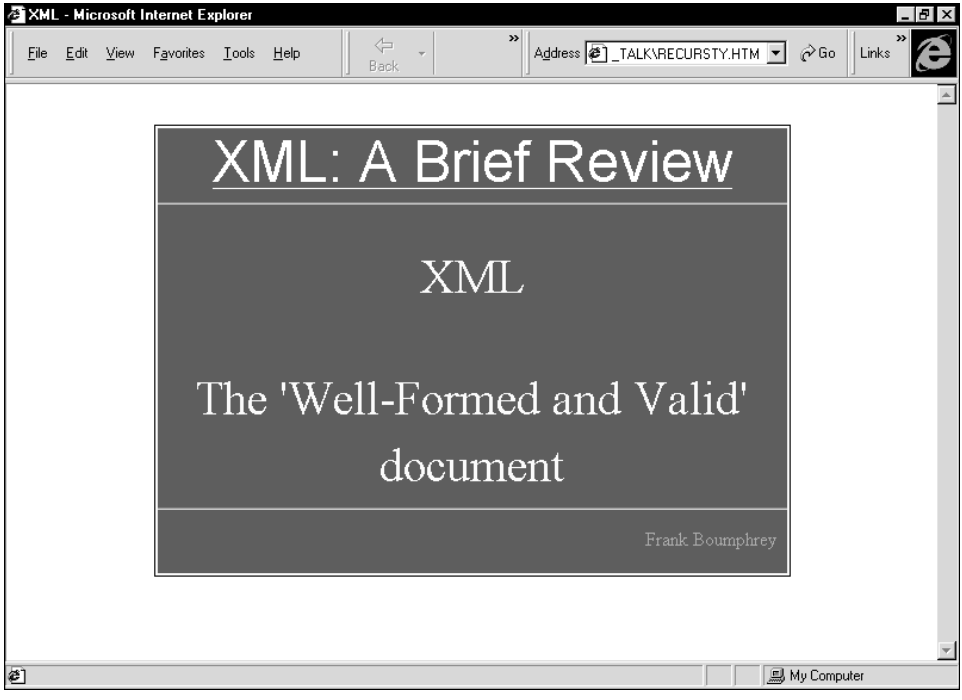


Рис. 6.22. Слайд с названием нового раздела

А вот код слайда с маркированным списком:

```
<slide>
<title>XML</title>
<slidebody>
   <header>XML: A Brief Review </header>
   <main>
   <line/>
   <bullet-list>
       <bullet_title>The valid document:</bullet_title>
       <item>Is well formed</item>
       <item>Conforms to its DTD</item>
   </bullet-list>

   </main>
   <footer>Frank Boumphrey</footer>
</slidebody>
</slide>
```

Сохраните этот код в файле `sl_cnet7.xml` и измените строку кода для островка XML в файле `slides.htm` на стоку:

```
<XML ID="island1" SRC="sl_cnet7.xml"></XML>
```

чтобы установить имя того слайда, который вы хотите продемонстрировать. Рис. 6.23 демонстрирует, как выглядит этот слайд в окне браузера.

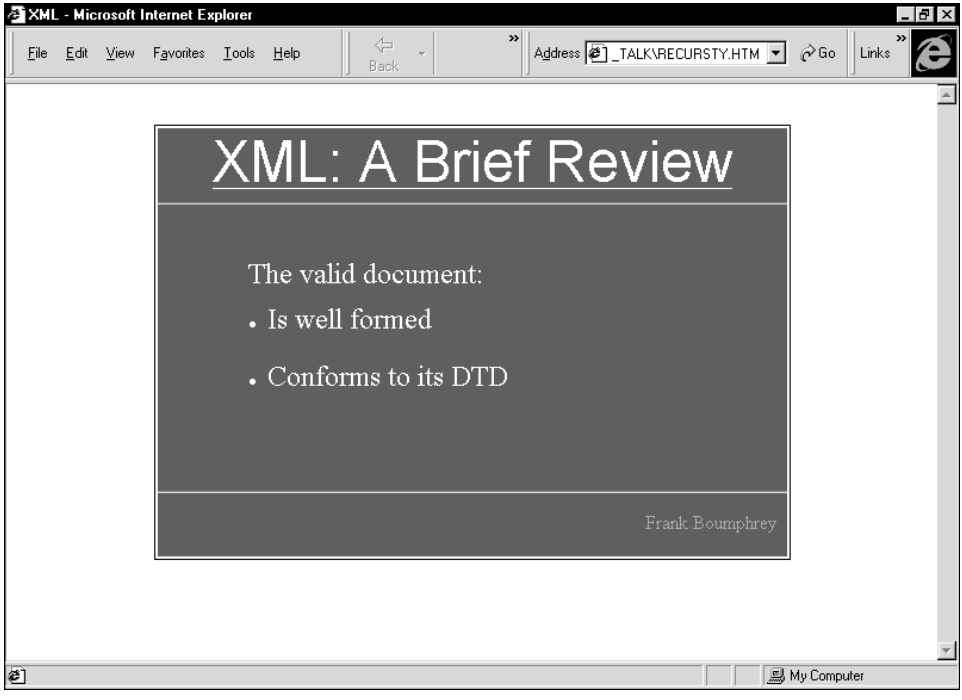


Рис. 6.23. Слайд с маркированным списком

Кажется, очень просто, но это живой пример. Он реален и был практически проверен на двух моих докладах.

Используя объектную модель документа, я смог поднять свои доклады на совершенно новый уровень интерактивности. Аудитория наблюдала за изменениями, производимыми в XML-слайде, и сразу же видела результат этих изменений на экране.

Другие примеры

По-моему, несмотря на всю свою простоту, приведенные примеры с успехом демонстрируют, что объектная модель документа может совершить революцию в использовании XML как клиентом, так и сервером. Применение DOM может быть ограничено только воображением пользователя.

XML и поисковые машины

Дотошный читатель в этом месте вправе спросить: «Задачи, о которых вы рассказали, вполне можно выполнить с помощью базы данных. Мои скрипты обшаривают базу данных и предоставляют информацию «на лету». Зачем мне хранить все данные в XML-файлах?»

Ответ состоит в том, что любая *поисковая машина* (data engine, search engine) или робот, посещающие ваш сайт, не имеют доступа к вашим данным и поэто-

му не в состоянии выполнить ничего полезного или осмысленного. Но если все данные размечены с помощью XML, робот способен распознать их и классифицировать.

Например, владелец ресторана может разметить набор своих услуг и предложений в соответствии со специальным определением типа документа, созданным ассоциацией ресторанов. Возможно, поисковому роботу дана задача найти подобные DTD, использовать объектную модель документа для извлечения нужной информации и выдать клиенту информацию, соответствующую его запросам.

Допустим, мне предстоит поездка в Чикаго, где придется отобедать в центре города с клиентом, который любит индийскую кухню. Я даю команду поисковому роботу за время моего перелета найти рестораны, удовлетворяющие этим требованиям. В Чикаго мне остается только подключить к сети мой портативный компьютер и познакомиться со списком подходящих заведений.

Правда, для этого должно существовать достаточное количество ресторанов, которые захотят разметить свои данные в соответствии с таким DTD.

Все это вполне возможно уже сейчас. Но никто так не поступает, поскольку нет данных, с которыми можно работать, то есть данных, размеченных в XML. Поэтому нет и стимулов создавать «разумный» поисковый робот. Следовательно, нет стимула размечать данные в XML. А поэтому нет стимула создавать «разумный» поисковый робот... Вечная проблема курицы и яйца.

Теперь, когда существует универсальная объектная модель документа, особое программное обеспечение для обработки XML-документов больше не нужно. Я думаю, что скоро увижу, как описанные выше способы применения набирают силу.

Заключение

В этой главе мы проанализировали применение объектной модели документа для XML.

Познакомившись с некоторыми общими концепциями объектной модели документа, мы перешли к модели, утвержденной консорциумом W3C, и разобрались, почему общепринятый интерфейс так важен для обработки XML-документов.

Мы подробно рассмотрели интерфейсы и использовали бета-версию браузера IE5 для демонстрации методов сборки.

В заключение было приведено несколько простых примеров реализации объектной модели документа, демонстрирующих ее огромный потенциал.

Глава 7. Просмотр XML–документов

Первоначальной целью, к которой стремились разработчики XML, являлось усовершенствование методов хранения, распределения и просмотра документов в Web, однако на практике возможности этого языка оказались намного шире. Сейчас его уже нередко рассматривают как средство хранения данных независимым от платформы и индивидуальных особенностей программного обеспечения способом.

Вообще говоря, чтобы иметь возможность работать с документами или воспользоваться какими-то данными, их прежде всего необходимо отобразить в приемлемой для восприятия форме. В этой главе как раз рассказывается, какого рода программы выполняют эти операции. В начале мы намерены обсудить следующие вопросы:

- в чем состоит различие при демонстрации XML- и HTML-документов и какие средства для этого применяются;
- что обычно понимают под таблицами стилей;
- природа потоковых объектов;
- другие способы просмотра содержания XML.

Далее будут рассмотрены:

- каскадные таблицы стилей (CSS) (они будут описаны так подробно, что вы сможете написать собственную таблицу стилей типа CSS);
- ситуация, сложившаяся в настоящее время, у двух основных производителей браузеров;
- трансформация и пользовательские процессоры;
- Spice – новый язык стилей.

В следующей главе мы продолжим эту тему и перейдем к введению в расширяемый язык таблиц стилей (XSL).

HTML в сравнении с XML

Как было сказано в начале, язык HTML 4 является приложением языка SGML, а следующее поколение HTML вероятнее всего станет приложением XML (информацию по этому вопросу можно найти на сайте http://www.w3.org/MarkUp/Activity).

Язык HTML состоит из конечного числа тэгов, которые все без исключения имеют некоторый смысл для соответствующего агента пользователя – в большинстве случаев эту роль выполняет браузер. Когда браузер встречает элемент `<H1>`, он «знает», что перед ним элемент для заголовка и показать его следует как блочный потоковый объект. Если браузер различает элемент `<I>`, он также «в курсе», что элемент необходимо продемонстрировать как встроенный потоковый объект, а содержащийся в нем текст должен быть изображен курсивом. Когда браузер видит тэг `<P>`, ему «известно», что все содержимое тэга вплоть до начала следующего блока, независимо от того, является ли следующий блок элементом `<P>` или каким-нибудь иным элементом, должно рассматриваться как единый блочный элемент.

Вскоре мы подробно расскажем о потоковых объектах, однако уже здесь важно дать исходные определения. *Потоковым объектом* (flow object) называется все, что при демонстрации документа «вытекает» на страницу. Каждый отдельный символ также является потоковым объектом и обладает собственным стилем, однако на практике обычно рассматриваются составные комплексы большего размера, такие как фразы, предложения или абзацы. Существует два основных типа потоковых объектов: *блочный потоковый объект* (block flow object), перед которым и после которого находится перевод строки, и *встроенный потоковый объект* (inline flow object), перед которым и после которого нет перевода строки.

Вот соответствующий раздел определения типа документа для языка HTML 4, описывающий элемент `P`:

```
<!ELEMENT P - O (%inline;)*           -- paragraph -->
```

В SGML конечные тэги не обязательны – это и означает буква `O`. Два дефиса `--` указывают, что и открывающий, и закрывающий тэги обязательны. При преобразовании HTML-документа в XML-документ все элементы `<P>` должны будут иметь закрывающий тэг.

Конечно, в XML каждый вправе создавать сколько ему заблагорассудится самых разнообразных тэгов, однако имейте в виду, что в результате никакое приложение не будет «знать», как их отображать. Поэтому для демонстрации XML какая-то таблица стилей абсолютно необходима. Наиболее вероятно, что знакомый с языком XML агент пользователя при встрече с XML-документом прежде всего начинает искать подобную таблицу. Если же ему это не удается, он сообщает о возникшем затруднении человеку, а затем вынужденно показывает все элементы как встроенные потоковые объекты.

Таблицы стилей

Таблица стилей представляет собой набор команд, указывающих, как следует изображать потоковый объект. Эти читаемые машиной команды, предназначенные для агента пользователя, могут, например, быть такими: «Взять данный отрывок текста и вывести его шрифтом Arial красного цвета размером 18 пунктов».

Таблицы стилей, доступные для человека

Существует несколько типов таблиц стилей, доступных для восприятия человеком. В этой главе мы довольно подробно рассмотрим *каскадные таблицы*

стилей (Cascading Style Sheets, CSS) и язык Spice, а в следующей познакомимся с *расширяемым языком таблиц стилей* (Extensible Stylesheet Language, XSL). Общим свойством всех этих методик является их способность быть читаемыми не только машиной, но и человеком. Другое преимущество подобных конфигураций проявляется при совместном их использовании с языками XML или HTML. Оно заключается в том, что одна и та же таблица стилей может быть применена к нескольким документам, обеспечивая тем самым некую модульную структуру.

Таблицы стилей, читаемые машиной

Очевидно также, что, кроме упомянутых, существуют немалое количество таблиц стилей, которые человек прочитать не в состоянии. Они записаны в двоичном формате, и большинство текстовых редакторов используют именно такие таблицы. Пользователь видит только результат. Когда, например, я выбираю стиль в редакторе Word 97, программа запрашивает свою двоичную таблицу стилей, каким образом следует форматировать текст. Мне остается только пользоваться готовым продуктом, а это не всегда удобно. Недостатки подобных таблиц, работающих в двоичном коде, очевидны:

- они не могут быть прочитаны человеком;
- при переносе их между платформами и приложениями нередко возникают проблемы.

Формат RTF

В этом разделе необходимо также упомянуть и о формате Rich Text Format. Он представляет собой внедренный в документ язык стилей в формате ASCII. Этот механизм независим от платформ и приложений, правда, человек тоже с большим трудом воспринимает его. Но главный недостаток формата RTF состоит в том, что, поскольку он содержится в самом документе, только для этого единственного документа его и можно использовать.

Потоковые объекты

Просмотр или записывание текстов или графических изображений на самом деле есть процесс создания особого рода *потоковых объектов* (flow objects). Наименьшим потоковым объектом, как уже было сказано, является одиночный символ или изображение (но не менее одного пикселя). Отдельные потоковые объекты могут быть сгруппированы в потоковые объекты большего размера, такие как фразы, абзацы, страницы или даже целые документы.

Применение стилей к потоковым объектам

Каждый потоковый объект по определению обладает тем или иным стилем, причем меньшие по размеру потоковые объекты, содержащиеся в нем, обычно оформляются в едином стиле, если на это счет нет никаких других специальных указаний. Таким образом, если выбрать в качестве потокового объекта абзац и применить к нему стиль, состоящий из шрифта Times New Roman черного цвета (вполне разумный выбор) размером 12 пунктов, то каждая буква и каждое слово здесь будут иметь один и тот же вид. В этом абзаце, однако, можно выбрать участок текста и применить

к нему другой стиль, например, использовать курсив. При этом первоначальный потоковый объект разделяется на четыре структурные единицы: первоначальный абзац и три дочерних потоковых объекта. К ним относятся: потоковый объект перед текстом, написанным курсивом; участок текста, оформленный курсивом; и еще один потоковый объект, следующий за участком, выделенным курсивом.

Потоковые объекты, находящиеся до и после участка курсива, обычно называются *анонимными потоковыми объектами* (anonimous flow objects).

Встроенные и блочные потоковые объекты

Выше (см. подраздел «HTML в сравнении с XML») было сказано, что потоковые объекты обычно делятся на два основных класса: *блочные потоковые объекты* (block flow object) и *встроенные потоковые объекты* (inline flow object). Блочным потоковым объектом называется участок документа, перед которым и после которого имеется перевод строки. Заголовок и абзац являются примерами таких потоковых объектов. В языке HTML содержание тэгов `<P>`, `<DIV>` и `<H1>` обычно форматируется как блочный потоковый объект.

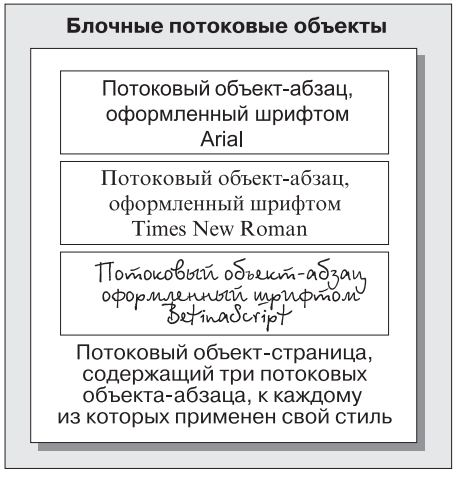


Рис. 7.1. Блочные потоковые объекты

На рис. 7.1 показаны три потоковых объекта в виде абзацев, расположенных на одной странице, к которым применены различные стили.

Встроенный потоковый объект – это потоковый объект, который появляется внутри строки текста. В этом смысле каждое слово в предложении является подобным объектом. Выделенную в тексте фразу также относят к этому классу. В HTML элементы `<I>`, `<B>` и `<SPAN>` обычно форматируются как встроенные потоковые объекты.

На рис. 7.2 показано, как при выделении курсивом части текста создаются три дочерних встроенных потоковых объекта.

Просмотр в браузере

Во время подготовки книги к изданию исходные коды браузера Netscape Mozilla 5, а также браузер Microsoft Internet Explorer 5 еще не поступили в продажу.

Бета-версии браузера IE5 демонстрируют хорошую поддержку XML, включенного в документ HTML. Мы довольно подробно рассмотрели этот вопрос в шестой главе, посвященной объектной модели документа.

Браузер Mozilla также обещает обеспечить полную поддержку XML с каскадными таблицами стилей (CSS). Однако по ранним бета-версиям трудно судить, в какой форме это будет сделано. При нехватке конкретной информации остается только предполагать, что поддержка таблиц стилей CSS совместно с XML будет обеспечиваться так же, как и в случае HTML.

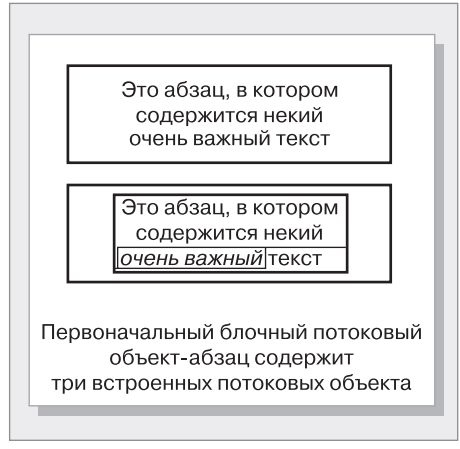


Рис. 7.2. Разбиение потокового объекта на дочерние объекты при выделении части текста

Хотелось бы добавить: ходят упорные слухи, что очень скоро все пользователи будут приятно удивлены конкретным решением по этому вопросу.

Способы демонстрации XML-файлов

Если у нас нет браузера для демонстрации XML, документ, выполненный в этой конфигурации, можно преобразовать в другой формат, например, в HTML. Скорее всего, большинство пользователей Web вплоть до 2000 года будут поступать именно таким образом.

Провести эту работу вручную довольно просто, правда, использование функции поиска и замены или простого скрипта может оказаться несколько утомительным делом. В этой главе мы рассмотрим простое приложение на Visual Basic, которое осуществляет преобразование в HTML.

Независимо от способа просмотра XML, пользователю необходимо иметь нечто вроде таблицы, которая будет сообщать программному обеспечению, какие стили применять к тем или иным объектам документа и как преобразовывать их в потоковые объекты.

Демонстрация на различных устройствах

В языке XML, в отличие от HTML, не делается никаких предположений, касающиеся структуры документа. По этой причине конфигурация XML особенно удобна для вывода документов на устройства, отличающиеся от монитора или печати. При использовании подходящей таблицы стилей и соответствующего программного обеспечения отобразить XML-документ на аппарате для чтения незрячими или на речевом синтезаторе будет так же легко, как и с помощью браузера. По тем же причинам XML удобен для вывода на устройства с низким разрешением.

Демонстрация приложениями пользователя

В связи с расширяющимся использованием XML для хранения данных, рассмотрим ряд XML-приложений, созданных для извлечения данных из XML-документа

и демонстрации их в виде записи или поля. Подобные операции сравнительно легко выполняются на языке Visual Basic 5, который обеспечивает доступ к базам данных.

Теперь, после краткого вступления, касающегося общих вопросов демонстрации документов XML, пора перейти к детальному разбору этой темы. Прежде всего давайте рассмотрим язык таблиц стилей CSS, который все шире и шире используется в Web.

Каскадные таблицы стилей

Сведения, помещенные в этом разделе, не претендуют на исчерпывающую полноту. Наша цель заключается в том, чтобы познакомить читателей с основами подобного метода. При этом авторы полагают, что после того как пользователи прочитают предлагаемый раздел, они вполне смогут справиться с написанием таблицы стилей для HTML- или XML-документа. Более полную информацию по этому вопросу можно найти в книге Professional Style Sheets for HTML and XML («Профессиональные таблицы стилей для HTML и XML»), изданной Wrox Press (ISBN 1-861001-65-7).

Большинство приведенных здесь примеров воспроизведены как на языке XML, так и на языке HTML, поскольку варианты, написанные на HTML, можно продемонстрировать на практике. Надеемся, что вскоре в пятых версиях браузеров будут работать оба типа учебных задач.

Что такое каскадная таблица стилей CSS

Таблица стилей CSS – это не более чем последовательность *правил стилей* (styling rules). В соответствии с ними, в первую очередь выбирается объект документа (обычно элемент), к которому нужно применить стиль, а затем уже описываются свойства этого стиля. Такая таблица должна вобрать в себя *правила стиля*, пробельные литеры и комментарии, более никаких других данных в ней быть не должно. Теперь рассмотрим одно из правил стиля и попытаемся разобраться, каким образом его можно использовать в согласии с XML- или HTML-документами.

Простое правило стиля CSS

Начнем с применения стиля к простому XML-документу и его HTML-эквиваленту. Вот простой документ `Hello_XML.xml`:

```
<xdoc>
<greeting> Здравствуй, XML! </greeting>
</xdoc>
```

В приведенной ниже таблице стилей существует только одно правило. Обратите внимание, что оно состоит из двух частей:

- *селектора* (selector), который указывает на элемент (в данном случае на элемент `<greeting>`), к которому нужно применить стиль;
- *описания* (declaration), которое указывает, какие свойства поставить в соответствие этому элементу, чтобы создать потоковый объект.

Таблица стилей хранится в отдельном файле `myxml_style.css`, и должна находиться в той же директории. Вскоре будет рассказано, как связать эти два файла, а сейчас давайте посмотрим на содержание файла с таблицей стилей.

Совет *Лучшим инструментом для создания таблицы стилей CSS авторы книги считают редактор NotePad (Блокнот) или аналогичный текстовый редактор. Как только вы напишете таблицу стилей CSS, сохраните ее в файле с расширением .css.*

```
greeting{
        display:block;
        font-size:16pt;
        color:red;
    }
```

Это правило указывает, что содержание каждого элемента `<greeting>` должно быть показано в виде блочного потокового объекта шрифтом красного цвета размером 16 пунктов. (Синтаксис мы обсудим немного позже, а пока имейте в виду: при использовании таблиц стилей CSS совместно с XML агенту пользователя необходимо сообщить, что он имеет дело с блочным потоковым объектом.)

Теперь рассмотрим эквивалентный HTML-документ:

```
<HTML>
<DIV> Здравствуй, XML!</DIV>
</HTML>
```

К нему тоже применяется правило стиля, находящееся в отдельном файле `myhtm_style.css`, который в нашем примере должен располагаться в той же директории, что и HTML-файл. Ниже приведено содержание файла с таблицей стилей:

```
DIV{
        font-size:16pt;
        color:red;
    }
```

Мы не сообщили HTML-процессору, как показывать элемент, поскольку он «знает», что все элементы `<DIV>` должны быть показаны в виде блочных потоковых объеков. Однако включение в файл строки `display:block;` не считается ошибкой. HTML-файл с этой строкой все равно можно просматривать XML-браузером, поэтому, вероятно, лучше всего сделать обязательной привычкой указание на способ форматирования во всех таблицах стилей для HTML.

Совет *Обратите внимание, что если бы мы вставили строку DIV {display:inline;}, она, возможно, была бы проигнорирована HTML-браузером, однако при просмотре документа в XML-браузере, элемент <DIV> был бы отформатирован как встроенный потоковый объект.*

Создав XML- и HTML-файлы, а также таблицы стилей, мы должны связать их друг с другом. Проблема состоит в том, как дать знать документу, где найти

таблицу стилей, которая содержит правила для его представления? Как только в действие будет введена связующая команда, к потоковому объекту `greeting/DIV` будет применен стиль со шрифтом красного цвета и размером 16 пунктов, который окажется шрифтом агента пользователя по умолчанию.

Совет *У агента пользователя обычно есть таблица стилей, используемая по умолчанию. Конечно же, все HTML-браузеры имеют такую таблицу стилей для HTML-документов. Если стиль не определен, применяется стиль по умолчанию.*

Соединение таблицы стилей и документа

Создание таблицы стилей – это только первый этап, далее необходимо таким образом присоединить ее к нашему документу, чтобы агент пользователя «знал», где найти правила стилей.

Присоединение HTML-документа

Существует два основных способа присоединения HTML-документа к таблице стилей: включение таблицы стилей в документ либо использование элемента `<LINK>`.

Правила стилей включаются в документ как содержание элемента `<STYLE>`:

```
<HTML>
<STYLE TYPE="text/css">
DIV{
       font-size:16pt;
       color:red;
   }
</STYLE>:
<DIV> Здравствуй, XML!</DIV>
</HTML>
```

Чтобы подсоединить документ к таблице стилей так называемым внешним способом, используется элемент `<LINK>`:

```
<LINK HREF="myhtml_style.css" REL="stylesheet"TYPE="text/css">
```

В окончательном виде документ с присоединенной извне таблицей стилей CSS имеет такой вид:

```
<HTML>
<LINK HREF="myhtml_style.css" REL="stylesheet"TYPE="text/css">
<DIV> Здравствуй, XML!</DIV>
</HTML>
```

Изображение на экране в обоих случаях должно быть одинаково.

Документы могут подсоединяться и другими способами, например, с помощью правила импортирования @import. Более полную информацию по этому вопросу можно найти в книге Professional Style Sheets for HTML and XML («Профессиональные таблицы стилей для HTML и XML»), изданной Wrox Press (ISBN 1-861001-65-7).

Присоединение HTML-документа описано и в спецификации CSS, ее следует искать по адресу http://www/w3.org/TR/REC-CSS2.

Присоединение XML-документа

Когда же приходится иметь дело с XML-документами, мы не можем поместить тэг стиля внутрь документа, и единственный способ передать информацию о присоединении и найти ей место в классическом прологе – это применить команду приложения:

```
<?xml:stylesheet href="myxml_style.css" type="text/css"?>
```

Уточнение *Последняя фраза не совсем верна, так как в такой ситуации можно использовать тэг пространства имен. В случае применения какой-либо схемы (например, DCD), инструкции стилей можно было бы поместить в нее. Однако в этой главе мы будем пользоваться только командой приложения.*

Окончательный вариант XML-документа выглядит следующим образом:

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

<?xml:stylesheet href="myxml_style.css" type="text/css"?>
<xdoc>
<greeting> Здравствуй, XML! </greeting>
</xdoc>
```

Результирующий вид файла на экране браузера должен быть одинаков во всех трех случаях (два варианта HTML-документа и один вариант XML-документа).

Присоединение XML-документа к таблице стилей рассматривается в заметке, которую можно найти на сайте http://www.w3.org/TR/NOTE-xml-stylesheet.

Мы забежали немного вперед, объясняя, как с целью создания потоковых объектов применять стили к объектам XML- и HTML-документов, а сейчас следует подробнее рассказать о правиле стилей CSS и его синтаксисе.

Правило стиля

Таблица стилей CSS, как было сказано в самом начале этого раздела, это не более чем совокупность правил стилей Cascading Style Sheets. Правила имеют одну и ту же основную конфигурацию, независимо от того, относятся ли они к элементу HTML или XML.

Основной синтаксис состоит из объекта или объектов, к которым нужно применить стиль, открывающей фигурной скобки, после которой следует свойство стиля, затем двоеточие и значение свойства. Конструкция завершается точкой с запятой, за которой следует другое свойство и так далее. Формально это выглядит так:

```
[объект (объекты), к которым применяется стиль]{([название свойства]:[значение
свойства];)+}
```

Первая часть правила называется *селектором*, поскольку она указывает на

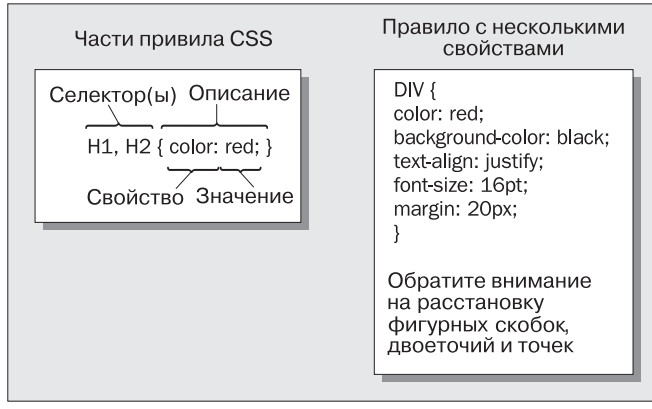


Рис. 7.3. Схема построения правила стиля

объект документа, к которому нужно применить стиль. Вторая часть, находящаяся внутри фигурных скобок, называется *описанием*, поскольку она описывает, какие войства нужно применить. Схема (рис. 7.3) поясняет этот синтаксис.

Обратите внимание, что последняя точка с запятой необязательна, однако на практике рекомендуется всегда ее ставить. Включение этого знака не является ошибкой, но если при добавлении правил стиля мы пропустим точку с запятой, то весь стиль будет считаться ошибочным.

Заглавные и строчные буквы

В HTML определение селектора как `DIV`, `div` и `Div` приведет к одному и тому же результату, поскольку язык HTML не различает заглавные и строчные буквы. Однако XML, как мы уже убедились, более строго относится к этим объектам, поэтому `Greeting` и `greeting` в этом языке будут двумя различными элементами. Другими словами:

```
Greeting{font-size:16pt;}
```

и

```
greeting{font-size:16pt;}
```

окажутся в XML двумя различными правилами.

Комментарии и пробельные литеры в CSS

Таблицы CSS используют комментарии в стиле старого языка C:

```
/*Все, что находится между открывающей косой чертой и звездочкой и звездочкой
и закрывающей косой чертой, является комментарием*/
```

Безусловно, комментарии не могут быть вложенными.

В большинстве случаев *пробельные литеры* (white spaces) браузером игнорируются, поэтому форматирование имеет значение только с точки зрения ясности отображения материала. В целях экономии места мы в этой книге, например, часто печатаем коды на каждой строке, но я настоятельно рекомендую для максимальной ясности писать код вразрядку.

Совет *Всякий, кому доводилось программировать, скажет, что ясность при написании кода является одним из самых важных условий успешной работы. То, что кажется абсолютно прозрачным сегодня, через несколько недель предстанет как некая загадка. Помните также, что, возможно, ваш код будет просматривать пользователь, который не знаком с ходом ваших мыслей. Необходимо разработать свой собственный понятный стиль и неукоснительно следовать ему. Если у вас есть сомнения по поводу ясности изложенного вами материала, или если причина, почему вы приняли то или иное решение, не совсем понятна, – вставьте комментарий. Это избавит вас от последующих бесконечных головных болей.*

Как упоминалось ранее, точка с запятой, которая используется для разделения свойств, необязательна, если имеется только одно значение. Однако лучше всегда ее ставить, поскольку опыт подсказывает, что может наступить момент, когда придется добавлять и удалять свойства, а отсутствие одной точки с запятой может привести к тому, что не будет показана вся страница. Когда дело доходит до таблиц стилей, браузер становится предельно требовательным к правильному синтаксису. Наиболее часто встречающиеся ошибки – это неправильное положение двоеточия или точки с запятой, а также пропуск обозначения единиц `pt` в строке `font-size:12pt`. Неправильное свойство браузер просто игнорирует, но часто он заодно пропускает все правило, иногда всю таблицу, а случается, что и целую страницу.

Свойства и значения

Описания свойств стилей и значения, которые они могут принимать, находятся в спецификациях CSS. В этой главе мы не ставим задачу подробно знакомить вас с этим документом. Свойства таблиц стилей CSS1 включены в «Приложение F», а свойства таблиц стилей CSS2 в «Приложение G»; в них также приводятся списки свойств, принимаемых ими значений, а также синтаксис и правила наследования.

С синтаксисом вы уже познакомились в примере:

```
[имя свойства]:[значение свойства];
```

Свойством является любое свойство стиля CSS. Значение может быть строкой, числом с единицей измерений, целым числом или значением цвета. Значения могут быть абсолютными или относительными, наследуемыми или нет.

Рассмотрим некоторые значения.

Единицы измерений

Существующие относительные единицы описаны в табл. 7.1.

Таблица 7.1. Относительные единицы в значениях свойств стилей

em	Высота шрифта элемента
ex	Высота буквы x
px	Пиксел. Его величина зависит от разрешения. Однако на самом деле спецификация CSS предлагает определенный размер пикселя и таким образом делает его почти абсолютной единицей
%	Процент

Допустимые абсолютные единицы приведены в табл. 7.2.

Таблица 7.2. Абсолютные единицы в значениях свойств стилей

in	Дюймы
cm	Сантиметры
mm	Миллиметры
pt	Пункт, определяемый как 1/72 дюйма
pc	Пика, определяемая как 12 пунктов

Абсолютные и относительные единицы

Примером абсолютной единицы является ширина поля страницы:

```
DIV{margin-left:2cm;}
```

В этом случае ширина поля будет 2 см независимо от размера экрана или страницы. В то же время запись:

```
DIV{margin-left:15%;}
```

говорит о том, что поле всегда будет составлять 15% от ширины страницы.

Наследуемые и ненаследуемые свойства

Некоторые свойства наследуются дочерними потоковыми объектами, а другие – нет. Следующий HTML-документ демонстрирует некоторые из них. Свойство `font-size` наследуется дочерним элементом `<P>` от родительского элемента `<DIV>`, но два других свойства – (`margin-left` и `background-color`) – не наследуются. Рассмотрим пример `07_inherit.htm`:

```
<HTML>
<STYLE TYPE="text/css">
   DIV{
       margin-left:2cm;
       font-size:16pt;
       border-style:solid;
       background-color:white;
   }
   P{
       border-style:solid;
   }
</STYLE>
<DIV>This text in a DIV is two centimeters from the left and 16 points.
   <P>
   So is this, there is no 'extra' margin for the P flow object. The font size is
inherited, but the margin isn't.
   </P
</DIV>
</HTML>
```

На рис. 7.4 показано, как файл `ch07_inherit.htm` выглядит в браузере IE5.

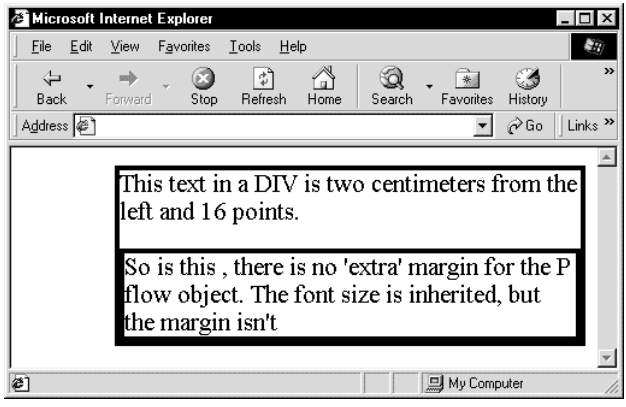


Рис. 7.4. Демонстрация наследования свойств

Как видим, свойство `font-size` наследуется, а свойство `left-margin` (левое поле) нет. Свойство `left-margin` для элемента `<P>` равно нулю или, другими словами, содержание элемента `<P>` выровнено по границе содержания элемента `<DIV>`. А как насчет свойства `background_color` (цвет фона)? Разве оно не наследуется? Нет, потому что значением по умолчанию для свойства `background_color` является значение `transparent` (прозрачный). Вследствие этого белый цвет фона элемента `<DIV>` «просвечивает» сквозь прозрачный фон элемента `<P>`.

Значения цветов

Значения цветов выражаются так же, как и в HTML, то есть:

```
#[шестнадцатеричное значение для красного] [шестнадцатеричное значение для зеленого] [шестнадцатеричное значение для синего]
```

Таким образом, `#888888` представляет собой светло-серый цвет, `#FFFFFF` – белый, а `#FF0000` – ярко-красный. Таблицы стилей CSS позволяют выражать цвета и в другом виде (табл. 7.3).

Таблица 7.3. Альтернативные способы задания значений цветов в CSS

	Серый	Белый	Красный
В сокращенном виде	#888	#FFF	#F00
Как десятичное целое значение	rgb(136,136,136)	rgb(255,255,255)	rgb(255,0,0)
Как значение, выраженное в процентах	Rgb(55%,55%,55%)	Rgb(100%,100%,100%)	Rgb(100%,0,0)

Формы правил каскадных таблиц стилей

Существует четыре основные формы, которые может принимать правило таблиц стилей CSS. Все они имеют одну и ту же структуру: `селектор {свойство: значение;}`. В табл. 7.4 показаны эти четыре типа и пример каждого из них.

Таблица 7.4. Основные формы правил каскадных таблиц стилей

Тип	Синтаксис	Пример
Простой селектор, одно описание	селектор1 {свойство1:значение1;}	BODY{color:red;}
Несколько селекторов или сгруппированные селекторы, несколько описаний	селектор2, селектор3 {свойство1:значение1; свойство2:значение2;}	H1,H2{font-size:32pt; color:#FF0000; text-align:center;}
Свойство с несколькими значениями	селектор4 {свойство1: значение1 значение2 значение3;}	P{font:italic bold 12pt Arial, sanserif}
Контекстный селектор	селектор5 селектор6 {свойство1:значение1; }	DIV EM{font-style:italic;}

Простой селектор

Это тот самый тип, который мы рассматривали в нашем первом примере (см. подраздел «Заглавные и строчные буквы»).

```
greeting{
color:red;
}
```

Содержание элемента `greeting` будет показано красным цветом.

Множественные селекторы

Разделяя элементы запятой, можно объединить несколько селекторов, как это сделано, например, в приведенном ниже примере:

```
greeting, farewell{
   display:block;
   color:red;
   font-family: 'times new roman',serif;
   font-size:16pt;
}
```

Как элемент `greeting`, так и элемент `farewell` будут рассматриваться в качестве блочных потоковых объектов, к которым применен стиль со шрифтом Times New Roman, красного цвета, размером 16 пунктов.

Свойство с несколькими значениями

Некоторые свойства могут принимать большое количество значений. В каких-то случаях это может быть удобно, но с точки зрения автора данной главы, подобное многообразие доставляет куда больше беспокойств, чем оно того стоит.

Свойство `font` – сокращение для `font-style` (начертание шрифта), `font-variant` (вариант шрифта), `font-weight` (вес шрифта), `font-size` (размер шрифта), `line-height` (высота строки), `font-family` (семейство шрифтов). Вот пример:

```
DIV{font:italic bold 12pt/24pt Times, serif}
```

Величина `font-variant` опущена. Если бы мы ее включили, то могли бы добавить значение `smallcaps` (малые заглавные буквы). Свойство `font-size` отделено от свойства `line-height` косой чертой. Таким образом, данная строка приведет

к выводу текста жирным курсивом, шрифтом Times размером 12 пунктов, через два интервала. (Или другим подходящим шрифтом с засечками, если Times отсутствует.)

Другими свойствами, обладающими различными значениями, являются `font`, `border`, `margin`, `padding`. Для уточнения деталей обратитесь к спецификации или справочной литературе.

Контекстные селекторы

Рассмотрим следующий XML-документ:

```
<xdoc>
   <heading><emphasis>Важный</emphasis> предмет</heading>
   <para>
       Докладчик обсудил несколько <emphasis>важных</emphasis> вопросов.
   </para>

   <para>
       Докладчик родом из Рима. <background>Римское гражданство <emphasis>важно</
emphasis> для каждого оратора.</background>
   </para>

</xdoc>
```

Элемент `<emphasis>` здесь встречается в трех случаях. В первый раз как дочерний к элементу `<heading>`, во второй – как дочерний к элементу `<para>` и в третий – как «внук» элемента `<para>` и дочерний к элементу `<background>`. *Контекстные селекторы* (contextual selectors) позволяют различать разные элементы `<emphasis>`. Употребление контекстного селектора в данном примере поясняется в табл. 7.5.

Таблица 7.5. Применение контекстных селекторов

emphasis {[описания]}	Выберет все элементы emphasis
heading emphasis {[описания]}	Выберет первый элемент emphasis
para emphasis {[описания]}	Выберет второй элемент emphasis
background emphasis {[описания]}	Выберет третий элемент emphasis
para background emphasis {[описания]}	Также выберет третий элемент emphasis, но с большей определенностью, чем в предыдущем случае

Конфликтующие правила

Что происходит, если группу конфликтующих правил применить к одному и тому же элементу, как это случилось в описанном выше примере? Для ответа на этот вопрос существует сложный набор правил, который можно найти в разделе 6.4.3 спецификации CSS2, называемом «Вычисление определенности селектора». При этом, однако, стоит помнить простейшее правило – выигрывает селектор, определенный наиболее точно. Придерживайтесь его и вы будете правы в 99% случаев. В списке селекторов, приведенном выше, мы продвигались от наименее определенного селектора к наиболее определенному.

Другие селекторы

В действительности существует много других способов, с помощью которых можно использовать синтаксис селектора, но они находятся вне рамок данной главы. Обратитесь за этой информацией к спецификации.

Каскадирование и наследование

Вопрос о наследовании свойств уже был кратко рассмотрен в подразделе, посвященном контекстным селекторам. Вообще говоря, наследование означает, что дочерний потоковый объект усваивает какие-то свойства своего предка. Единственный способ проверить, какие свойства наследуются, а какие нет – это обратиться к «Приложению F» и «Приложению G» или к соответствующей спецификации. При этом опять-таки следует придерживаться простейшего правила: если по смыслу свойство может наследоваться, то скорее всего оно наследуется.

Что имеется в виду под словом «каскадные», употребленным в названии таблиц стилей? Для ответа на этот вопрос рассмотрим следующий XML-документ:

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

<?xml-stylesheet href="myxml_style.css" type="text/css"?>
<?xml-stylesheet href="yourxml_style.css" type="text/css"?>
<?xml-stylesheet href="hisxml_style.css" type="text/css"?>
<?xml-stylesheet href="herxml_style.css" type="text/css"?>
<?xml-stylesheet href="itsxml_style.css" type="text/css"?>
<?xml-stylesheet href="ourxml_style.css" type="text/css"?>
<?xml-stylesheet href="theirxml_style.css" type="text/css"?>

<xdoc>
<greeting> Hello XML! </greeting>
</xdoc>
```

Здесь у нас представлено не менее семи таблиц стилей, и во всех может быть упомянут элемент `greeting`. Какой же стиль победит при просмотре этой записи браузером?

Ответ на этот вопрос осуществляет сложный механизм, описание которого можно найти в спецификации (раздел 6.4.1 – «Порядок каскадирования»), однако есть и более простое правило. Оно гласит: если никакая таблица стилей не отмечена атрибутом `!important` (о чем будет сказано в следующем разделе), то преимущество имеет последняя таблица. Другими словами, правило проходит по каскаду сверху вниз.

Описания !important

Свойство как `!important` (важное) можно описать следующим образом:

```
P{font-weight:bold !important}
```

Характеристика, отмеченная атрибутом `!important`, имеет преимущество перед любым другим свойством, не обладающим подобным атрибутом.

Вообще говоря, в спецификации CSS2 авторские правила стилей имеют преимущество перед любыми пользовательскими правилами стилей, но если и автор, и пользователь описывают одно и то же свойство как `!important`, то преимущество имеет пользователь.

Совет *Пользователь может описать свою собственную таблицу стилей в браузере IE5, используя пункты меню Tools ⇒ Internetoptions ⇒ General ⇒ Accessibility (Сервис ⇒ Настройки Internet ⇒ Общие ⇒ Доступ). В браузере IE4 настройки Internetoptions находятся в разделе View (Вид).*

В этом состоит концептуальное отличие спецификаций CSS1 (первый уровень) и CSS2 (второй уровень). В спецификации CSS1 все было наоборот. Если немного подумать, то правомерность такого изменения станет очевидной, поскольку пользователь может иметь определенные затруднения (например, с различением цветов), которые требуют других стилей.

Кратко познакомившись с наследованием и каскадированием, рассмотрим, как таблица стилей CSS обрабатывает блочные потоковые объекты.

Рамки

Рамка (box) – это термин, который используется в таблице стилей CSS для описания относящихся к этой структуре потоковых объектов. В действительности рамки являются *контейнерами* (containers) для текста и элементов. Мы не без умысла упомянули слово «контейнеры», имея в виду, что рамки вполне укладываются в это понятие, однако в таблице стилей CSS им было присвоено несколько упрощенное название.

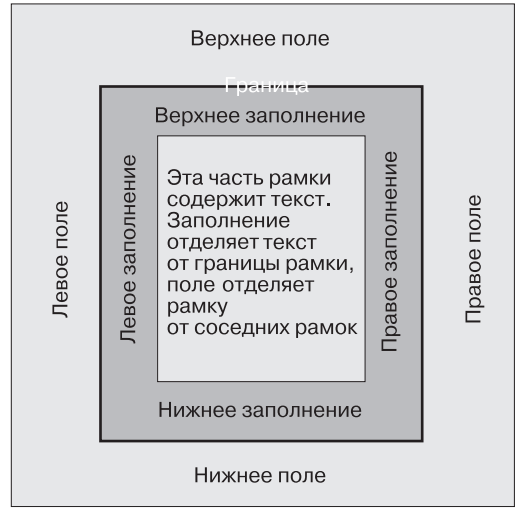


Рис. 7.5. Схема конструкции рамки

Рамки имеют множество свойств, и это не удивительно. Любой блок текста может быть помещен в рамку, а она, в свою очередь, ищет место на *основе* (canvas), то есть в окне браузера. К этой конфигурации применятся три основные свойства, а именно: `margin` (поле), `border` (граница) и `padding` (заполнение) (рис. 7.5).

Каждая рамка прежде всего имеет *границу* (border) – порой невидимую, – которая отделяет ее от края основы или от соседних рамок. Пространство между границей и внешним краем соседней рамки или между границей и содержащей рамкой, назвается *полями* (margin). Пространство между содержимым рамки и ее границей принято называть *заполнением* (padding).

Заметим, что в примере на рис. 7.6 в вертикальном направлении все поля схлопываются так, что видно только большее поле. Другими словами, вместо добавления полей и образования значительного расстояния между рамками, они просто отделяются друг от друга более широким из двух полей. В горизонтальном направлении сохраняются оба поля.

Вот простой пример, находящийся в файле `ch07_box.htm`, который вы можете загрузить или запустить с нашего сайта `http://webdev.wrox.co.uk/books/1525`:

```
<HTML>
<HEAD>
```

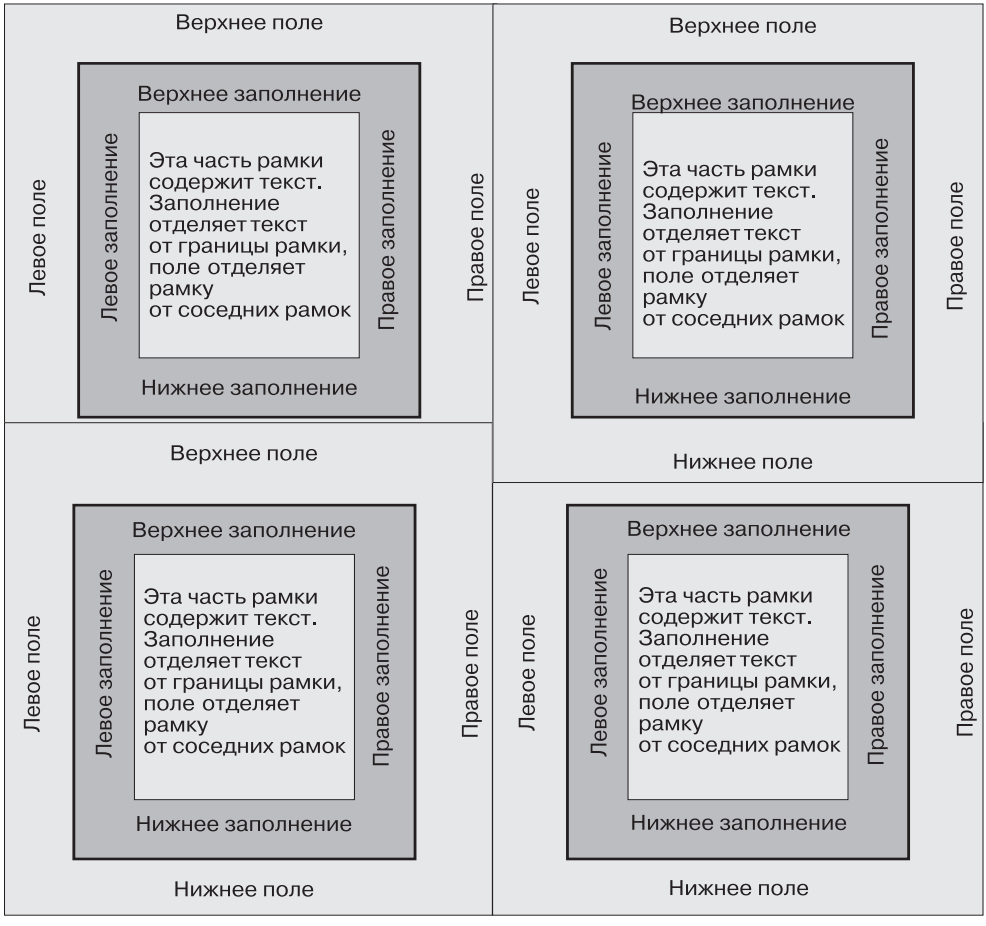


Рис. 7.6. Взаимодействие нескольких рамок

```
<TITLE>Рамки</TITLE>

<STYLE TYPE="text/css">
    P{
        font: 14pt 'Times New Roman',serif;
        color:black;
        margin:0.5in;
        padding:0.25in;
        border:solid blue;
    }

    DIV{
        font: italic bold 12pt Arial,sans-serif;
        color:red;
        margin:0.25in;
        padding:0.25in;
        border:solid green;
```

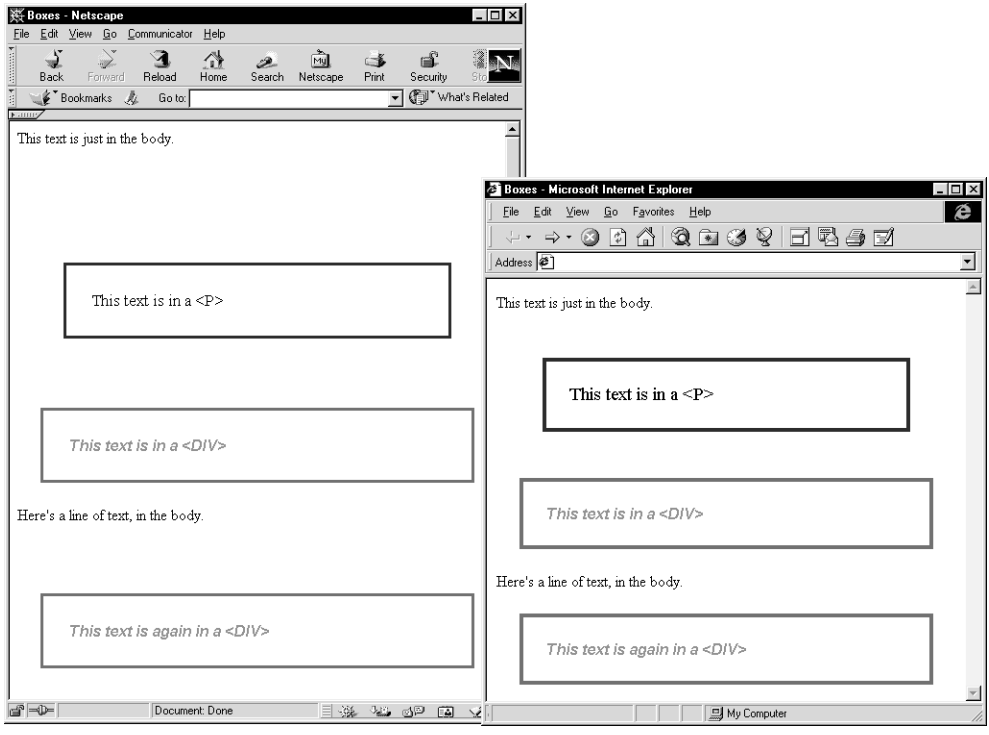


Рис. 7.7. Интерпретация рамок браузерами IE4 и Communicator 4

```
    }

    EM{
        color:red;
    }

    DIV EM{
        color:green;
        font-style:normal;
    }

</STYLE>
</HEAD>

<BODY>
This text is just in the BODY.

<P>
This text is in a &lt;P&gt;.
</P>

<DIV>
This text is in a &lt;DIV&gt;.
</DIV>

Here's a line of text in the body.
```

```
<DIV>
This text is again in a &lt;DIV&gt;.
</DIV>

</BODY>
</HTML>
```

Этот пример иллюстрирует одну из основных проблем, серьезно затрудняющую работу с таблицами стилей. Суть ее состоит том, что браузеры Communicator 4 и IE4 совершенно по-разному интерпретируют этот код, что и показано на рис. 7.7.

Браузер IE4 демонстрирует код правильно (почти). Текст окружен аккуратной рамкой. Между текстом и границей имеется заполнение шириной в одну четвертую дюйма. Кроме того, рамка окружена полями в ту же одну четвертую дюйма. А вот в браузере Communicator 4 поля рамок не схлопываются.

В браузере IE4 там, где рамки примыкают друг к другу в вертикальном направлении, поля схлопываются друг с другом таким образом, что пространство между содержанием элементов `<P>` и `<DIV>` оказывается 0.5 дюйма, а не 0.5+0.25 дюйма.

Совет *Предупреждение. Если вы вставляете текст из системы обработки текстов в простой текстовый редактор, то в нем иногда появляется символ цитирования, отличающийся от традиционных кавычек ASCII, код 34. В этом случае ваш код не будет работать. Поэтому если вы решили использовать систему обработки текстов, внимательно проследите за одинарными и двойными кавычками.*

Классы

Одно из самых значительных достижений таблиц стилей CSS состоит в том, что они позволяют определять ваши собственные *классы стилей* (classes of styles), а точнее, классы селекторов. Полное описание этого способа достаточно обширно и сложно для понимания, но основы его просты, так что давайте попытаемся разобраться с ними в той мере, в какой это необходимо, чтобы вы могли в своей работе пользоваться классами. Это понятие порой приобретает особое значение в HTML, поскольку именно эти характеристики позволяют программному обеспечению отличать разные структуры.

Рассмотрим диалог из пьесы Шекспира «Сон в летнюю ночь»[1]:

Гермия. Я хмурю бровь – он любит все сильней.
Елена. Такую власть – улыбке бы моей!
Гермия. Кляну его – в нем только ярче пламя!
Елена. О, если б мне смягчить его мольбами!
Гермия. Чем жестче я, тем он нежней со мной.
Елена. Чем я нежней, тем жестче он со мной!
Гермия. В его безумье – не моя вина.
Елена. Твоей красы! О, будь моей, – вина!

[1] Текст приводится в соответствии с переводом Т. Щепкиной-Куперник. Полн. собр. соч.: В 8 т. М., 1958. Т. 3. (*Прим. ред.*)

Если бы появилась необходимость различить текст каждого из действующих лиц в HTML-документе, необходимо поставить в соответствие как Гермии, так и Елене особые классы, и решить задачу таким образом:

```
DIV.hermia{font-style:italic; color:blue;}

DIV.helena{font-style:normal; color:black}
```

Обратите внимание, как мы выполнили эту операцию: просто добавили точку, за которой следует имя. Оно должно состоять из букв и цифр. Включение дефисов (-) тоже допустимо, но имя не может начинаться с цифры или дефиса.

Теперь при создании HTML-файла мы можем использовать атрибут `CLASS`:

```
<DIV CLASS="hermia">Гермия. Я хмурю бровь – он любит все сильней.</DIV>
<DIV CLASS="helena">Елена. Такую власть – улыбке бы моей!</DIV>
```

Если хотите посмотреть, как это выглядит на практике, используйте файл `ch07_play.htm` на нашем сайте http://webdev.wrox.co.uk/books/1525.

Заметим, что имя класса размечено семантически, поскольку это свойство позволяет нам не просто описывать стиль, что было бы стилистической разметкой, но в какой-то мере отражать содержание, указывая, где слова Елены, а где слова Гермии. Понятно, что в приведенном примере в этом нет необходимости, просто здесь предложено хорошее упражнение для читателя. Теперь можно использовать атрибут `CLASS` для осмысленного поиска в HTML-документе. Например, можно найти все строки Гермии с помощью поиска `CLASS="hermia"`.

Таким образом, можно придать языку HTML некоторые семантические преимущества, присущие XML.

Обратите внимание вот на какое обстоятельство. Поскольку язык HTML в отличие от XML не различает заглавные и строчные буквы, возьмите за правило записывать все тэги HTML заглавными, а все тэги XML – строчными буквами. (Это соглашение предлагается также в языках XSL и XML.)

Каким способом можно выполнить ту же самую задачу разделения текстов в XML? Ниже приведена версия XML использованного выше примера, размеченная Йоном Бозаком (Jon Bosak).

```
<speech>
<speaker>hermia</speaker>
<line>Я хмурю бровь – он любит все сильней.</LINE>
</speech>

<speech>
<speaker>helena</speaker>
<line>Такую власть – улыбке бы моей!</LINE>
</speech>

<speech>
<speaker>Hermia</speaker>
<line>Кляну его – в нем только ярче пламя!</LINE>
</speeCh>

<speech>
```

```
<speaker>Helena</speaker>
<line>О, если б мне смягчить его мольбами!</LINE>
</speech>
```

Следует упомянуть еще об одном способе разделения текстов. Речь идет об использовании атрибута-селектора. (Все типы селекторов здесь обсуждаться не будут, без комментариев будет также описан синтаксис. За более полной информацией следует обращаться к спецификации или к разработкам для специалистов.) Например, XML-документ можно изменить (с помощью хотя бы скрипта) следующим образом:

```
<speech>
<speaker NAME="HERMIA"/>
<line>Я хмурю бровь – он любит все сильней.</LINE>
</speech>

<speech>
<speaker NAME="HELENA"/>
<line>Такую власть – улыбке бы моей!</LINE>
</SPEECH>
```

а затем записать такую контекстную таблицу стилей:

```
SPEAKER [NAME=HELENA] LINE {color:blue;}

SPEAKER [NAME=HERMIA] LINE {color:black;}
```

Заметим, что значения атрибутов в таблицах стилей CSS *не заключаются* в кавычки!

Другой способ состоит в использовании атрибута `CLASS` и пространств имен. Этот способ имеет дополнительное преимущество, так как позволяет применять атрибут `STYLE` языка HTML.

Пространства имен и атрибуты CLASS и STYLE

Пространства имен уже были рассмотрены в четвертой главе. Если описать это пространство HTML в корневом элементе документа, содержащем выдержку из комедии Шекспира «Сон в летнюю ночь», используя:

```
<PLAY xmlns:HTML="http://www.w3.org/HTML">
<TITLE>A Midsummer Night's Dream</TITLE>
```

то в этом случае можно использовать атрибуты `CLASS` и `STYLE` так, как они могли бы быть применены в HTML, то есть:

```
<speech>
<speaker>HERMIA</SPEAKER>
<line HTML:STYLE="color:blue;">Я хмурю бровь – он любит все сильней.</LINE>
</speech>

<speech>
<speaker>HELENA</SPEAKER>
<line HTML:STYLE="color:black;">Такую власть – улыбке бы моей!</LINE>
</SPEECH>
```

Это особенно удобно, когда необходимо использовать одну и ту же таблицу стилей для демонстрации XML и HTML версий документа в тождественном виде. Конечно же, это предполагает, что наш агент пользователя распознает пространства имен HTML. (Учтите, никакой из них вплоть до версии 4 не способен на это!)

Использование пространств имен и XML в HTML-документе

Для объяснения, как пространства имен применяются в HTML-документе, мы воспользовались ранней бета-версией браузера IE5. Следующий пример демонстрирует, как использовать стили в тэге XML, включенном в HTML-документ:

```
<HTML>

<xml:namespace prefix="myns"/>

<style>
.mystyle1{font-size:24pt;display:block}
</style>

<BODY BGCOLOR="white" id="body">
   <P>An example of an using a namespace to apply style to an XML tag</P>

   <xtag1>The default display property for an unknown tag is inline.</xtag1>
   <xtag2>As is demonstrated here</xtag2>

   <myns:xtag class="mystyle1">An XML tag with style applied.</myns:xtag>
   <myns:xtag class="mystyle1">A display='block' property has been applied.</
myns:xtag>

<P>The following script demonstrates that the 'namespace tag' is treated as a
known HTML tag.</P>
<SCRIPT>
   var x=document.all.length
   for(var i=0;i<x;i++)
       {
       document.write(document.all[i].tagName+ " ... ")
       }
</SCRIPT>
</BODY>
</HTML>
```

Если вы запустите этот пример в браузере IE5, то увидите окно экрана, приведенное на рис. 7.8. Заметим, что тэги XML с пространствами имен рассматриваются как тэги HTML, в то время как тэги без указания пространств имен рассматриваются как неизвестные. Об этом свидетельствует включение в дерево документа закрываюих тэгов. Вы можете загрузить код или запустить пример на нашем сайте http://webdev.wrox.co.uk/books/1525. Файл называется `ch07_names.html`.

Обратите внимание, как тэги `<HEAD>` и `<TITLE>` «подразумеваются» объектной моделью документа HTML.

На этом завершается обсуждение таблиц стилей CSS. Оно оказалось кратким, однако, на наш взгляд, достаточно подробным, чтобы разобраться, каким образом

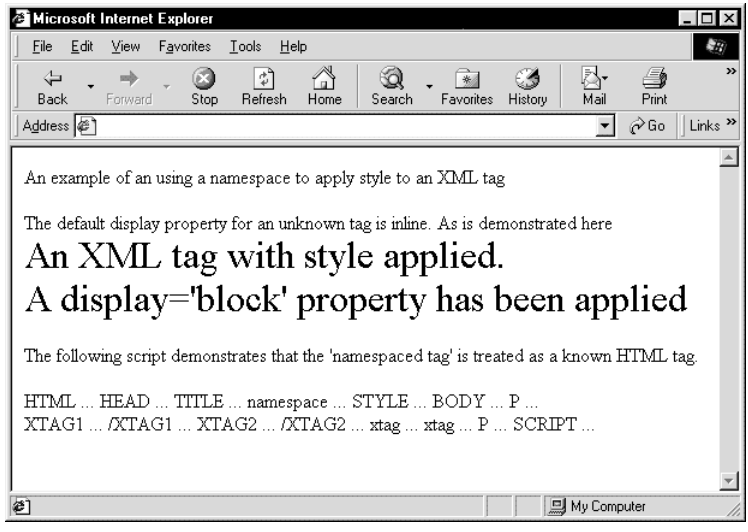


Рис. 7.8. Пример использования пространства имен

таблицы стилей CSS в большинстве случаев удовлетворяют потребности программистов в применении стилей. Еще одно преимущество таблиц стилей CSS заключается в том, что значительная часть спецификации CSS1, а также спецификации CSS2 успешно реализованы, и скорее всего в еще большем объеме будут отражены в пятых версиях обоих браузеров.

Теперь рассмотрим некоторые преобразования XML, предоставляющие возможность просматривать XML-документы с помощью агентов пользователя, не совместимых с XML.

Преобразование XML-документов

Наряду с просмотром XML-документов в браузере, можно преобразовать правильный документ в другое определение ипа документа, например в HTML, а уж затем просматривать преобразованный документ. Этот процесс иллюстрируется схемой, приведенной на рис. 7.9.

Преобразование вручную

Об этом способе часто забывают, несмотря на то, что когда возникает необходимость просмотреть всего один-два документа, его применение оказывается вполне приемлемым. Документ вручную трансформируется в HTML, затем к нему присоединяется таблица стилей для просмотра. Как выполнить эту операцию, вновь продемонстрируем на примере обработки отрывка из пьесы Шекспира «Сон в летнюю ночь».

Документ загружается в текстовый редактор с добротной функцией **Search and Replace** (Найти и Заменить), например, Windows WordPad. Затем необходимо систематически пройти по всем тэгам, заменяя каждый открывающий тэг `<SPEECH>` тэгом `<DIV CLASS="SPEECH">`, а каждый закрывающий тэг `</SPEECH>` тэгом `</DIV>`.

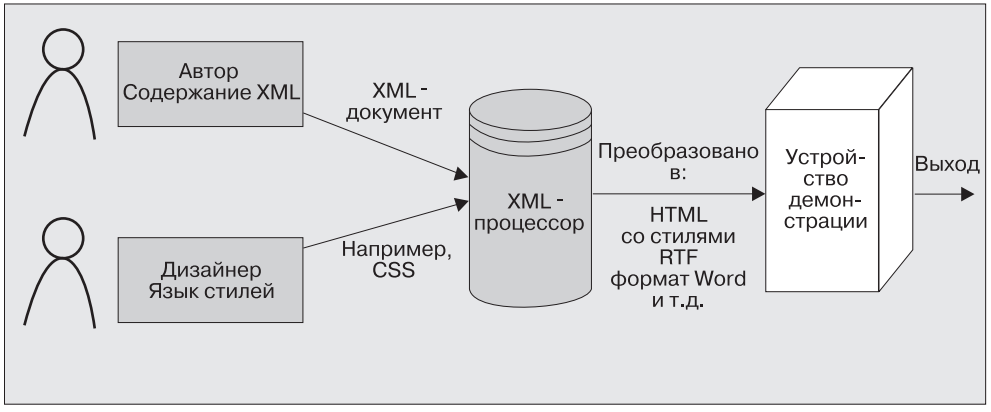


Рис. 7.9. Демонстрация XML-документа путем преобразования в другие форматы

На рис. 7.10 показано, как в этом случае выглядит экран.
Вот фрагмент файла `dream.xml`:

```
<speech>
<speaker>HELENA</speaker>
<line>Такую власть – улыбке бы моей!</LINE>
</SPEECH>

<speech>
<speaker>HERIA</speaker>
<line>Кляну его – в нем только ярче пламя!</LINE>
</speech>
```

Далее приводится этот же фрагмент после трансформации его в `dream.htm`.

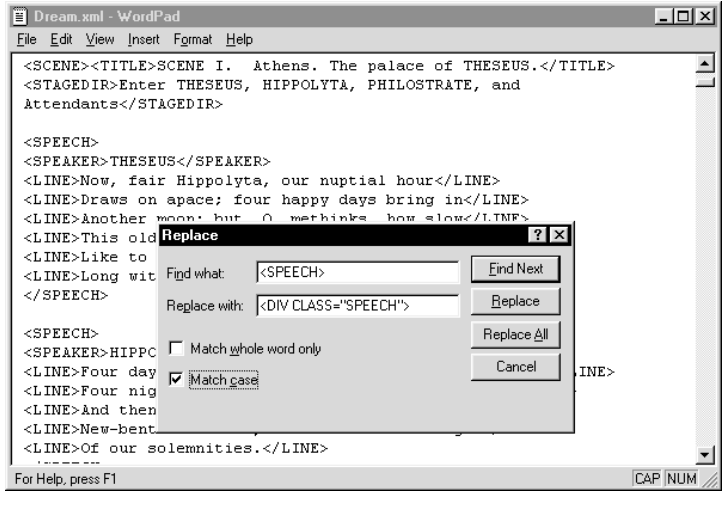


Рис. 7.10. Ручное преобразование документа из XML в HTML

```
<div class="speech">
<div class="speaker">helena</div>
<div class="line">Такую власть - улыбке бы моей!</DIV>
</DIV>
<div class="speech">
<div class="speaker">hermia</div >
<div class="line">Кляну его - в нем только ярче пламя!</DIV >
</DIV>
```

Весь процесс занимает около пяти минут, что делает его вполне удобным для просмотра небольшого количества документов.

Использование анализатора XMLparse.exe

Анализатор `XMLparse.exe` – это достаточно простой инструмент, написанный автором на Visual Basic в целях обучения. Он позволяет открыть XML-документ, проверить, является ли он состоятельным, применить к нему стиль и преобразовать в HTML для демонстрации. Чтобы запустить файл `XMLparse.exe`, загрузите его с Web-сайта издательства Wrox http://webdev.wrox.co.uk/books/1525 и поместите в ту же директорию, что и XML-файлы, с которыми вы предполагаете его использовать. К программе `XMLparse.exe` прилагается файл `Readme.txt`, в котором приведена информация по установке.

Совет *Обратите особое внимание, что эта программа всего лишь инструмент для обучения, который не годится для сколько-нибудь серьезной обработки XML-документов.*

Окно программы изображено на рис. 7.11.

В качестве примера для обработки выберем один из файлов с контактной информацией, для которого в предыдущей главе мы создали определение типа документа. Все файлы, пригодные для такого случая, можно загрузить с Web-сайта издательства Wrox: http://rapid.wrox.co.uk/books/1525.

Итак, мы остановились на XML-файле `ch07_contacts.xml`.

```
<client>
<name id="CPQ142">
<honorific> Dr.</honorific>
<first> Pierre</first>
<middle> R. </middle>
<last> LeBlanc </last>
<nickname> Butch</nickname>
</name>
<phone> 440-123-4567</phone>
<company lang="french"></company>
<contact type="first" ><date> Jan 1992</date></contact>
<contact type="last"><date> Dec 19 1997</date></contact>
<attitude interest="warm"/>
<personal> Baby: Girl b. <date> Nov 1997</date>, golf mad!, handicap 7, likes
Mexican food, completely bilingual Frenchand English</personal>
</client>
```

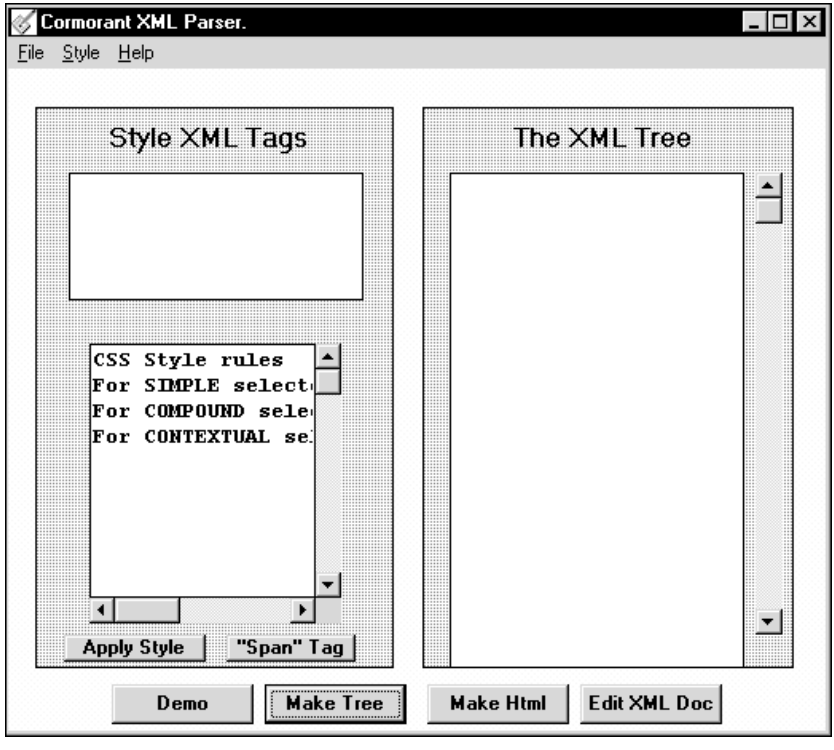


Рис. 7.11. Окно программы XMLparse.exe

Прежде всего, используя команду **Open** (Открыть) в выпадающем меню **File**, откроем документ в анализаторе (рис. 7.12).

Как видите, документ обрабатывается автоматически, и в правом окне, называемом **XML Tree** (Дерево XML), возникает дерево документа.

В верхнем левом окне появляется список всех тэгов, а также простые инструкции, как создать объединенный или контекстный селекторы.

Щелчок правой кнопкой мыши на любом открывающем тэге дерева приводит к открытию в отдельном окне XML-файла с выделенным соответствующим тэгом (рис. 7.13).

Чтобы применить стили к документу, следует прежде всего решить, какие из элементов будут определены как встроенные элементы. Чтобы сделать элемент встроенным, достаточно щелкнуть по его имени в окне со списком, а затем нажать кнопку **Span Tag**. Заметим, что по умолчанию элемент считается блочным.

Допустим, что все тэги с именами и с датами являются встроенными. Тогда, чтобы применить стиль к тэгу, щелкаем левой кнопкой мыши по соответствующему элементу в окне **Style XML Tags** (применить стили к тэгам XML), и в окне внизу появляется селектор с незаполненным описанием. На рис. 7.14 показано, как элементу `personal` присваивается описание:

```
{font-size:16pt;color:navy;}
```

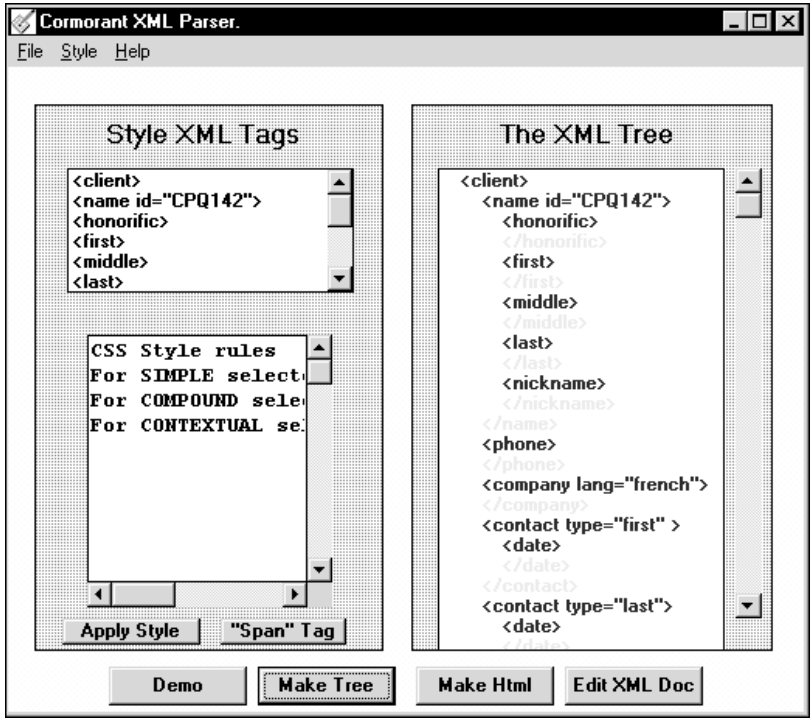


Рис. 7.12. Открытие XML-документа в программе XMLparse.exe

Как только правило введено в место для описания, следует щелкнуть по кнопке **Apply Style**, чтобы применить стиль. Это ознаает, что при преобразовании XML-документа в HTML все, содержащееся в тэгах `<personal>`, будет показано шрифтом размером 16 пунктов темно-синего цвета.

Повторим описанный выше процесс применения стилей так, чтобы:

- фамилия (last name) была выведена жирным шрифтом;
- прозвище (nickname) было выведено красным цветом;
- номер телефона и личная информация были выведены шрифтом размером 16 пунктов;
- из элемента `<date>` дочернего к элементу `<personal>` создать контекстный селектор и выделить дату красным цветом.

Чтобы преобразовать XML-документ в HTML, необходимо использовать кнопку **Make HTML**. Анализатор создает файл `temp.htm`, содержащий код HTML. При этом открывается окно редактора Notepad (Блокнот) с кодом (рис. 7.15). Код можно исправить по желанию, а затем использовать кнопку **Save As** меню, чтобы сохранить его в виде HTML-файла. Сохраним этот файл под именем `ch07_contacts.htm`.

В окне редактора Notepad показана часть таблицы стилей, а также тэги HTML `<DIV>` и `<SPAN>` с соответствующими атрибутами `CLASS`, в которые были преобразо-

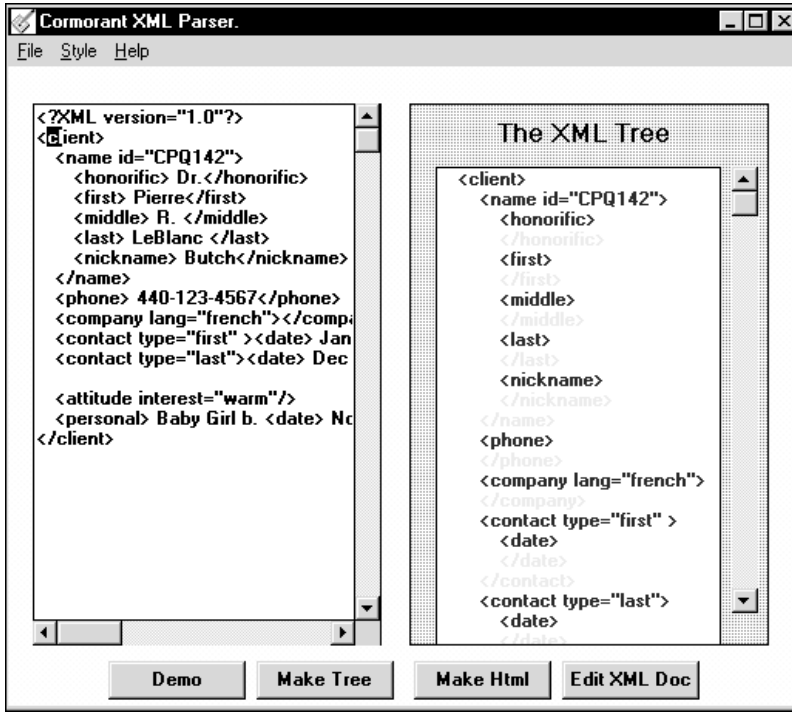


Рис. 7.13. Переход к указанному тэгу в программе XMLparse.exe

ваны тэги XML. На рис. 7.16 показано, как выглядит преобразованный XML-файл в брузере Netscape Communicator.

Как видите, на этом очень просто примере становится вполне понятно, что такое современный спосо использования XML.

Преобразование XML со «старой» таблицей стилей XSL

Компания Microsoft разработала интересный анализатор, запускаемый из командной строки, который преобразовывал XML-документ и таблицу стилей XSL «старого» типа в HTML.

По сведениям на октябрь 1998 года анализатор все еще можно найти по адресу

http://www.microsoft.com/xml/xsl/downloads/msxsl.asp

а учебное пособие по его использованию – на сайте

http://www.hypermedic.com/style/xsl/xsl_tut1.txt

Этот анализатор, служивший главным образом для предварительного ознакомления с технологией подобного процесса, использовался для преобразования бесчисленного количества XML-файлов. Возможно, по этой причине таблицы стилей XSL стали так популярны. В одиннадцатой главе для применения стилей к приложению «Туристический маклер» используется эта же программа.

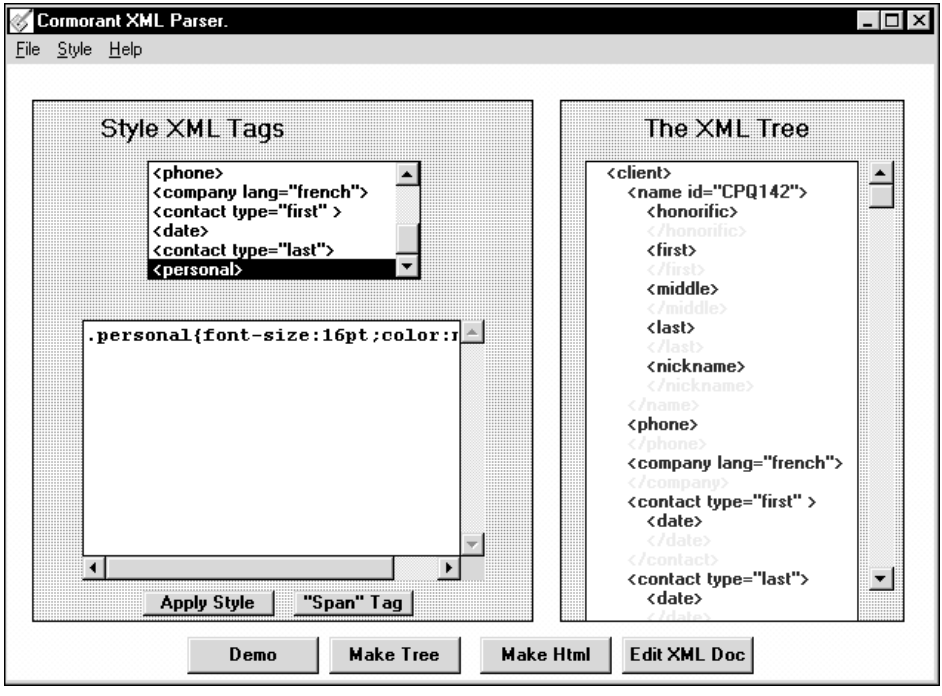


Рис. 7.14. Применение стиля к тэгу в программе XMLparse.exe

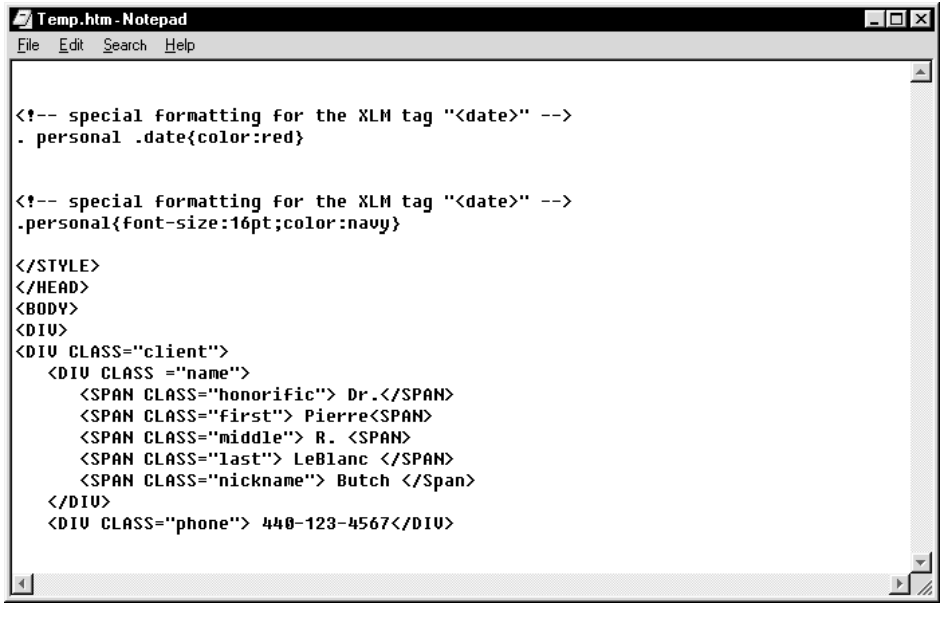


Рис. 7.15. Файл, преобразованный в HTML, в окне редактора Notepad

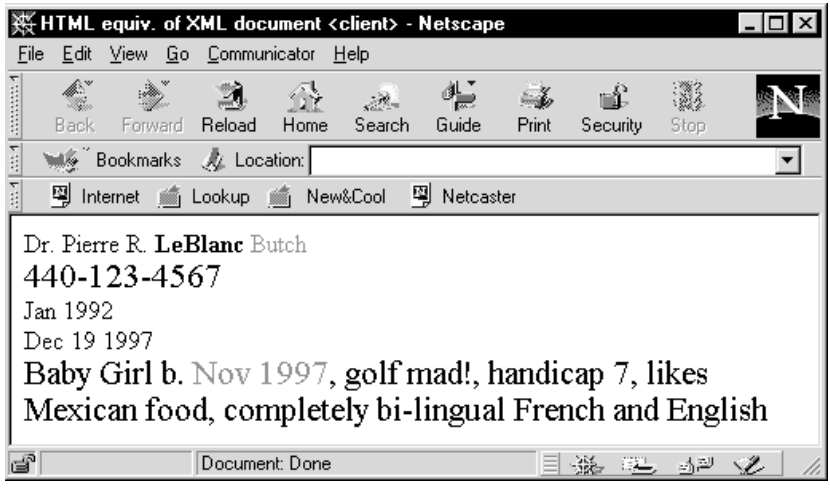


Рис. 7.16. Просмотр XML-файла, преобразованного в HTML

Язык Spice

Spice – это новый язык стилей, написанный Дэйвом Рэггетом (Dave Raggett) и Робертом Стиваном (Robert Stevahn) из компании Hewlett Packard. Если программист в достаточной мере владеет таблицами стилей CSS и языком JavaScript, применение этого средства многократно увеличивает эффект обработки данных, к тому же Spice может быть использован для применения стилей как к XML-, так и к HTML-документам. Аналогично языкам XSL и DSSSL, но в отличие от языка таблиц стилей CSS, этот метод позволяет обрабатывать потоковые объекты и даже разрешает пользователю создавать свой собственный тип потокового объекта в виде макроса. Несмотря на то, что во время подготовки книги к изданию средства для чтения таблиц стилей Spice еще не появились на рынке, но к осени 1998 года ожидался выпуск специального браузера и дополнительного программного обеспечения к основным браузерам, позволяющего читать таблицы стилей Spice.

В этой главе мы довольно подробно рассмотрим Spice, поскольку в настоящее время информации о нем не хватает. Однако учтите, что в настоящее время описываемый язык еще находится на стадии формирования, поэтому все приведенные здесь сведения вполне могут измениться. Таким образом, данный раздел можно рассматривать только как введение в Spice.

Концепции языка Spice

Для такого мощного языка, как Spice, его концепции выглядят удивительно просто. Основной конструкцией является правило стиля, которое записывается с использованием свойств типа CSS. Ниже приведен пример такого правила:

```
style H1
{
   fontFamily: "Arial";
```

```
    fontSize: 12pt;
    display: block;
}
```

Есть еще несколько важных моментов, на которые следует обратить внимание:

- свойства `font-family` и `font-size` записаны не через дефис, а «верблюдом» (горбом верблюда служит заглавная буква во втором слове) во избежание путаницы с оператором вычитания языка JavaScript;
- в тексте описан блочный потоковый объект `block` (блок). Объект `block` можно найти в библиотеке потоковых объектов, поддерживаемой машиной Spice. Этот механизм можно отыскать по определенному адресу, на который можно сослаться с помощью URL (более подробная информация будет приведена позже);
- определения потоковых объектов должны быть специально импортированы (см. следующий раздел).

Ниже показан один из способов, с помощью которого правила Spice могут быть включены в HTML-документ.

```
<HTML>
<STYLE TYPE= "text/spice">

import document, block, inline;
    import wroxheader from "http://www.wrox.com/Spice/std.lib"
    /*Написанное выше приведено только для примера!*/
    import "Mystyle.css"
    import "Hisstyle.spi"
    style HTML
        {
            fontFamily: "Times New Roman";
            fontSize: 12pt;
            display: block;
        }
    style H1
        {
            fontFamily: "Arial";
            fontSize: 1.5em;
            display: wroxheader;
        }
    style H2
        {
            fontFamily: "Arial";
            fontSize: 1.5em;
            textAlign: center;
            display: block;
        }
    style P
        {
            textAlign: left;
            display: block;
        }
    style EM
```

```
    {
        fontStyle: italic;
        display: inline;
    }
</STYLE>
    <H1> 15 Spice and XS</H1>
    <H2> Spice</H2>
    <P> Hello Spice!</P>
    <P> This is a<EM> really cool</EM> language.</P>
</HTML>
```

На рис. 7.17 показано, как выглядел бы этот файл в браузере, поддрживающем язык Spice. Следует заметить, изображение изготовлено искусственно, только для того, чтобы показать, на что должен быть похож подобный файл.

Рис. 7.17. Просмотр документа на языке стилей Spice в гипотетическом браузере

На что следует обратить внимание при изучении Spice

Мы включили правила стилей в тэги `<STYLE>`, но их, вероятно, можно будет включать и в тэги `<SCRIPT>`. В результате пользователи получат ряд преимуществ, поскольку элемент `<SCRIPT>` допускает атрибут `SRC`, который можно использовать для включения любого модифицирующего кода. (Дальнейшую информацию вы найдете в разделе «Потоковые объекты языка Spice».)

Также как и CSS, таблицы стилей Spice сконструированы по каскадному принципу. В примере выше импортированы как каскадные таблицы стилей, так и таблицы стилей типа Spice. (В таблице стилей CSS мы бы применили оператор `@import`, а здесь используем просто `import`.) Скоро мы обсудим это подробнее.

Также как и каскадные таблицы стилей, таблицы стилей Spice имеют возможность наследовать. В дополнение к установленному стилю `textAlign:left;` содержание элемента `<P>` будет воспроизведено с импортированными стилями `document`, а именно `fontFamily: "Times New Roman"; fontSize:12pt;` поскольку элемент `P` содержится в блоке `document`.

Обратите внимание на оператор `import` в заголовке документа. Он является полным аналогом оператора `import` в языке Java или оператора `include` в языке C и импортирует библиотеку функций. В данном случае это набор функций, которые сообщают воспроизводящему устройству, как показывать потоковые объекты. Подробный разговор на эту тему состоится далее.

Этот простой пример создает шесть потоковых объектов: `HTML`, `H1`, `H2`, два объекта `P` и объект `EM`. Они будут показаны в соответствии с определениями, данными в импортированных библиотеках.

Может показаться, что язык Spice всего лишь усложненный способ написания таблиц стилей CSS. Однако его реальная сила обнаруживается в том, как он обрабатывает потоковые объекты, а также в тех случаях, когда приходится создавать новые потоковые объекты. Но сначала следует немного разобраться, каким же образом Spice воспроизводит документы.

Воспроизведение документов

Когда агент пользователя получает таблицу стилей Spice, он последовательно проходит по родительскому документу и по очереди исследует каждый элемент. Затем он ищет правило, которое может применить к элементу, и создает на его основе потоковый объект. Если агент не находит правила, он применяет правило по умолчанию (`default`).

Потоковые объекты языка Spice

Spice может использовать любой тип потоковых объектов, если только они включены в библиотеку. Этот перечень может быть жестко встроен в агент пользователя, находиться в кэше либо во внешнем URL, на который можно сделать ссылку.

В примере, рассмотренном выше, тэг `<H1>` воспроизводится в виде потокового объекта `wroxheader`. Но агент пользователя еще должен найти код, описывающий, как должен воспроизводиться подобный тип потокового объекта. Прежде всего, агент погружается в свою встроенную библиотеку, затем в кэш, на предмет того, не загружался ли недавно этот код, а затем отправляется в URL и импортирует нужный код. Если все эти операции оказались напрасными – пользователь может быть не подключен к сети – агент воспроизводит потоковый объект по умолчанию.

Чтобы не показывать объект, следует использовать свойство `display:none;`.

Модификация потокового объекта

Spice также позволяет создавать наши собственные потоковые объекты. Их можно произвести заново или воспользоваться модификацей уже существующего потокового объекта. Чтобы осуществить этот процесс, следует обратиться к языкам Java, JavaScript, самому Spice; можно даже создать объект ActiveX.

Вот пример того, как, модифицируя блочный потоковый объект с использованием Spice, можно сформировать новый потоковый объект `wroxheader`:

```
// Отметим сходство с ключевым словом 'extends' языка Java

   prototype wroxheader extends block
   {
```

```
        function layout(element)
        {
            this.style.borderStyle=solid;
            this.style.backgroundColor=black;
            this.style.color=white;
            this.style.position=absolute;
            this.style.top=0;
            this.style.left=0;
            this.append (new Text("Chapter "));
            processChildren(element,this);
        }
    }
```

Обратите внимание на использование ключевого слова `prototype`, которое извещает воспроизводящее устройство Spice о том, что мы создаем новый прототип и называем его `wroxheader`.

Заметим также, как используется ключевое слово `extends`. Оно обявляет воспроизводящему устройству Spice, какой именно объект мы модифицируем для создания прототипа `wroxheader`. Подобный прием очень похож на использование этого ключевого слова в языке Java.

Теперь при использовании пары свойство-значение `display:wroxheader;` мы получим белый текст на черном фоне в левом верхнем углу страницы, и перед текстом будет стоять слово Chapter (глава). Таким образом, в нашем предыдущем примере:

```
<H1>13 Spice и XS</H1>
```

будет воспроизведено шрифтом Arial размером 1,5*12 = 18 пунктов (наш базовый размер шрифта – 12 пунктов) в соответствии с инструкциями таблицы стилей. В соответствии с инструкциями по демонстрации данного потокового объекта, текст белого цвета появится в левом верхнем углу страницы на черном фоне.

Напомим соответствующий участок таблицы стилей.

```
style H1
    {
    fontFamily: "Arial";
    fontSize: 1.5em;
    display: wroxheader;
    }
```

Вид текста на экране будет аналогичен рис. 7.18.

Рис. 7.18. Модификация объекта в языке стилей Spice

Поместим созданный код в файл `std.lib`, на который, как мы видели, есть ссылка в начале нашей таблицы стилей.

В коде, приведенном выше, встречается несколько ключевых слов языка Spice. Функции `layout` передается имя элемента, к которому нужно применить формат

(то есть H1). Ключевое слово append используется для добавления нового дочернего объекта в последовательность, в данном случае дочернего объекта "Chapter" типа text. Метод processChildren является встроенным методом, который проходит по дереву, обрабатывая дочерние объекты элемента. (В нашем примере дочерних объектов нет.) Наряду с добавлением текста, язык Spice позволяет присоединять графику из стандартной библиотеки. (См. ниже в разделе «Графика».)

Теперь рассмотрим, каким образом Spice позволяет управлять последовательностью воспроизведения в нашем документе.

Режимы и непоследовательное воспроизведение

Также как и XSL, язык Spice позволяет пользователям организовывать режимы и осуществлять непоследовательное воспроизведение. Разберем простой XML-документ:

```
<document>

   <contents></contents>
   <chapter> Introduction</chapter>
   <chapter_text> In this chapter....</chapter_text>
   <chapter> CSS</chapter>
   <chapter_text> Cascading Style....</chapter_text>
   <chapter> Spice</chapter>
   <chapter_text> Spice is a new...</chapter_text>

</document>
```

Наша задача – выбрать все названия глав и напечатать их в тэге contents (оглавление), воспользовавшись стилем, отличным от того, который был применен в заголовках.

Как это сделать?

При работе с таблицами стилей CSS программисту пришлось бы переписать документ и применить стили к заголовкам по отдельности.

```
<document>

<contents>
   <chapter class="toc"> Introduction</chapter>
   <chapter class="toc"> CSS</chapter>
   etc, etc
</contents>
   <chapter class= "heading"> Introduction</chapter>
   <chapter_text> In this chapter     </chapter_text>
   etc, etc
```

Здесь все названия глав включены в элемент <contents>, и при использовании с этим элементом заголовкам присвоен класс toc, а при использовании в начале каждой главы – класс heading. Таким образом, необходимо просто сослаться на таблицу стилей CSS, которая содержит следующие правила:

```
.toc{font-size:10pt}
.heading{font-size:18pt}
```

В языке Spice можно найти более эффективный способ выполнить эту операцию с помощью правила mode. Первый шаг состоит в создании правила стиля для элемента <chapter> в целом:

```
style chapter
    {
    fontSize:18pt;
    }
```

На следующем этапе организуется специальный режим для демонстрации глав в виде оглавления:

```
mode toc
    {style*
    {
        display:none;
    }
    style chapter
    {
        fontSize:10pt;
    }
}
```

Обратите внимание на то, что мы сделали. Звездочка сообщает процессору о том, что правило display:none (не демонстрировать) нужно применить к каждому элементу, поэтому не будут показаны никакие элементы, кроме элемента <chapter>, который будет воспроизведен шрифтом размером 10 пунктов. Теперь, чтобы просмотреть содержимое элемента <contents>, присваиваем тэгу в качестве атрибута уникальный идентификатор id. Например:

```
<contents id= "toc"></contents>
```

Когда процессор встречает такую запись, он создает потоковый объект, в котором содержимое каждого элемента <chapter> воспроизводится шрифтом размером 10 пунктов.

Заметим, что если бы мы не включили в наше правило записи:

```
style*
{
    display:none;
}
```

то вся книга была бы воспроизведена в оглавлении!

Другой способ решения данной задачи состоит в создании прототипа для потокового объекта, называемого, например, body.

```
prototype body extends block
{
    // Форматирует содержание книги.
    with mode toc   // Просто демонстрирует элементы chapter.
        {
        processChildren (element,this)
        }
```

```
        // Теперь документ отформатирован должным образом.

        processChildren(element,this)
}
```

Теперь, при использовании следующего правила стиля в примере, рассмотренном выше:

```
    style document
        {
        Display:body;
        }
```

документ будет отформатирован таким образом, что в его начале появится оглавление книги. Обратите внимание, что нам даже не пришлось использовать тэг `<contents>`. Spice предоставляет кратчайший способ сделать то же самое следующим образом:

```
    style document
        {
        Display: block with mode toc, block with mode regular
        }
```

Более подробную информацию можно найти в сообщениях и учебных пособиях, ссылки на которые приведены в «Приложении B».

Таблицы стилей, зависящие от системы воспроизведения

Язык Spice позволяет указать, какую систему воспроизведения использовать в том или ином случае. Например, задача состоит в том, чтобы часть демонстрации озвучить, а часть – показать. Для этой цели используется правило `media`.

```
media aural
{
    style body
    {
        volume: medium;
        voiceFamily: male;
    }

    style abbr
    {
        volume: medium;
        voiceFamily: female;
    }
```

В результате все содержимое, к которому применен стиль `body`, будет воспроизведено мужским голосом, а та его часть, к которой применен стиль `abbr`, будет «озвучена» женским голосом. Вот как можно использовать это определение в прототипах с правилом `media aural`:

```
with media aural
{
```

```
        ProcessChidren(element,this);
    }
```

для создания потоковых объектов с речевым выводом.

Графика

При создании прототипа `wroxheader`

```
prototype wroxheader extends block
```

было использовано ключевое слово `append`:

```
this.append (new Text("Chapter"));
```

для добавления текста к потоковому объекту. Можно было бы также создать ссылку на графический объект, используя URL или стандартную библиотеку. Например:

```
this.append (new Graphic("Wroxlogo"));
```

или

```
this.append (new Graphic("http:/www.wrox.com/graphics/logo.gif"));
```

Присоединение таблиц стилей Spice

В приведенном в самом начале примере было показано, как для участия в каскадировании импортировать одну таблицу стилей в другую. Таким же образом таблицу стилей Spice можно поставить в соответствие XML- или HTML-документу.

Чтобы указать таблицу стилей, следует использовать стандартную команду приложения языка XML. Например:

```
<?xml-stylesheet href= "docstyle.spi" type= "text/spice"?>
```

В HTML используется тэг `LINK`:

```
<LINK REL= "stylesheet" href= "docstyle.spi" type= "text/spice">
```

Уровень разработанности языка Spice

В настоящее время Spice находится на стадии сообщения (представленного 3 февраля 1998 года), и, насколько известно, консорциум W3C пока не проявляет особой активности в этом вопросе. Тем не менее есть надежда, что соответствующий браузер появится уже в ближайшем будущем, и поскольку язык Spice хорошо подходит для Active X controls и аплетов Java, с уверенностью можно сказать, что его введут туда вместе с другими языками стилей, которые будут использоваться для оформления XML-документов.

Заключение

В этой главе мы коснулись вопросов просмотра XML. Главный вывод заключается в том, что для демонстрации элементов, относящихся к этому языку, необходимо предоставлять таблицу стилей, так как они, в отличие от элементов HTML, не встроены в браузер.

Напомним, что существует два способа просмотра XML: преобразование XML в другой формат, например, HTML; либо просмотр напрямую с использованием таблиц стилей. В настоящее время прямая демонстрация применяется редко, поскольку браузеры не оказывают достаточной поддержки этому языку. Однако с появлением пятых версий браузеров положение должно решительно измениться.

Далее мы познакомились с двумя различными языками стилей, а именно: каскадными таблицами стилей (CSS) и Spice. В следующей главе будет рассмотрен язык стилей XSL.

Глава 8. Расширяемый язык таблиц стилей XSL

В этой главе мы кратко расскажем о новой спецификации языка XSL от 18 августа 1998 года. Прежняя спецификация была основана на сообщении, опубликованном в октябре 1997 года, а также на простенькой программе `msxsl.exe`, созданной компанией Microsoft. Эта программа преобразовывала документы XML вместе с таблицами стилей XSL «старого» типа в документы HTML со встроенными в них стилями. Это был очень популярный, и, возможно, простейший способ просмотра XML-документов, доступный и в настоящее время.

Совет *Подробную информацию о том, как использовать эту программу, можно найти в книге Professional Style Sheets for HTML and XML («Профессиональные таблицы стилей для HTML и XML»), изданную Wrox Press, ISBN 1-861001-65-7, или на сайте http://www.hypermedic.com/style.*

Программу `msxsl.exe` можно загрузить по адресу http://www.microsoft.com/xml и использовать в сочетании с таблицами стилей XSL «старого» типа, как описано в первоначальном сообщении.

В этой главе будет рассмотрен язык XSL следующего поколения. К сожалению, для «нового» XSL еще не существует средства, сравнимого с `msxsl.exe`, поэтому данная глава как бы попадает в слишком хорошо известную приверженцам XML категорию статей, смысл которых заключается в извержении восхищений: «Как замечательно сделано! Вот будет здорово, когда появится подходящая программа!..» (Однако во время составления сборника авгуры от Microsoft во всеуслышание заявляли о серьезной поддержке «нового» XSL в последней бета-версии Internet Explorer 5.)

В дополнение к этому следует заметить, что во время подготовки книги августовский рабочий проект спецификации XSL во многом был еще «в процессе работы», а в решении некоторых вопросов попросту зияли обширные пробелы.

В спецификации также открыто говорится, что в ней не определяется механизм, с помощью которого таблица стилей XSL должна быть связана с XML-документом. В то же самое время в спецификации утверждается, что одним из возможных механизмов организации этого процесса может быть указание ссылки на таблицу стилей XSL с помощью команды приложения внутри XML-документа, примерно так, как это было сделано для таблиц стилей CSS. Таким образом, в случае с XSL

команда приложения могла бы выглядеть следующим образом:

```
<?xml:stylesheet href="mystyle.xsl" type="text/xsl"?>
```

Однако следует помнить, что во время составления книги этот вопрос все еще являлся предметом обсуждения.

Невзирая на все предостережения, совершенно очевидно, что язык XSL создан для того, чтобы стать одной из основных частей XML-технологии, и эта глава предоставит вам всю необходимую информацию для быстрого освоения его ожидаемых в скором будущем реализаций.

О чем говорится в этой главе

Эта часть сборника не претендует на то, чтобы считаться исчерпывающим описанием XSL. Наша цель состоит лишь в предоставлении предварительных – «рабочих» – знаний о «новой» спецификации XSL и информации, достаточной для того, чтобы разобраться в этом документе, изучить который, предупреждаем сразу, нелегкий труд. Это, впрочем, касается и всех прочих материалов, тоже находящихся на стадии обсуждения. Надеемся, что данная глава поможет вам преодолеть эти трудности.

В этой главе мы рассмотрим:

- общие концепции языка XSL;
- построение *результирующего дерева* (Result Tree) из *исходного дерева* (Source Tree);
- форматирование и применение стилей к результирующему дереву.

Начнем с краткого описания основ процесса применения стилей с помощью таблиц стилей XSL.

Краткий обзор

Как известно, XML-документ состоит из некоторого количества объектов. В полном объеме этот вопрос обсуждался в предыдущей главе, так что в данный момент нас будут интересовать только те объекты, которые представлены в виде элементов, а также текстовые объекты.

Язык XSL создает дерево документа, состоящее из потоковых объектов. Затем он применяет стиль к этим объектам и выдает их агенту пользователя для обработки. Весь процесс можно разделить на три части:

- из исходного дерева первоначального XML-документа строится дерево потоковых объектов (в терминах XSL – результирующее дерево);
- к каждому узлу результирующего дерева применяются правила стилей;
- агент пользователя демонстрирует результирующее дерево, используя соответствующие стили.

XSL-документ состоит из определенного числа *правил шаблона* (template rules), которые имеют две части: *образец* (pattern) и *действие* (action). (Все упомянутые термины будут вскоре объяснены.)

Образец правила выбирает в первоначальном документе объект, из которого будет создан *потоковый объект* (flow object) для результирующего документа.

Действие правила описывает, какой тип потокового объекта создавать и какой стиль к нему применять.

Как только создание результирующего дерева завершено, дерево передается агенту пользователя для демонстрации.

Ниже приведен упрощенный пример, иллюстрирующий этот процесс. Не стоит пока обращать внимание на синтаксис, в первую очередь попытайтесь разобраться с общими концепциями.

Здравствуй, XSL!

Вот пример правильного XML-файла:

```
<greeting>Здравствуй, XSL!</greeting>
```

Ниже приведен XSL-файл. Он довольно сильно упрощен, поскольку мы договорились обращать внимание только на основную идею. Для ясности, например, опущена информация о пространстве имен в начальном тэге `xsl:stylesheet` (см. раздел «Пространства имен и таблицы стилей XSL»).

```
<xsl:stylesheet>

        <!--Это часть-образец.-->

    <xsl:template match="greeting">

        <!--Это часть-действие.-->

        <fo:block color="red" font-size="16pt">
            <xsl:apply-templates/>
        </fo:block>
    </xsl:template>
</xsl:stylesheet>
```

Здесь видно, как элемент `xsl:template` сопоставляется с элементом XML-файла. Это образец правила шаблона.

Соответствие устанавливается с помощью атрибута `match`. В исходном XML-документе выбирается элемент `<greeting>`, из которого создается потоковый объект для применения стиля.

После того как объект выбран, мы переходим к части *действие* правила.

Прежде всего, необходимо указать, какой тип потокового объекта мы хотим создать. В данном случае это блочный потоковый объект, что и фиксируем с помощью элемента `<fo:block>`. О потоковых объектах поговорим в следующем разделе, но по сути блочный потоковый объект – это потоковый объект, до и после которого имеется перевод строки. (Префикс пространства имен `fo:` мы рассмотрим позже.)

Стили к потоковому объекту применяются с помощью атрибутов. Заметим, что эти атрибуты идентичны свойствам таблиц стилей CSS. При использовании данной методики следовало бы написать `color:red`. В результате преобразования этого свойства в атрибут стиля XSL получаем `color="red"`.

Далее, используя элемент <xsl:process-children/>, обрабатываем дочерние объекты элемента <greeting>. У элемента greeting есть только один дочерний объект – текстовый узел "Здравствуй, XSL!", который будет передаваться агенту пользователя для демонстрации в виде блочного потокового объекта, выведенного красным шрифтом в 16 пунктов.

Итак, элемент xsl:template содержит информацию о форматировании и стиле. Элемент greeting будет отформатирован как блочный элемент (то есть до и после соответствующего потокового объекта будет стоять перевод строки). Кроме того, мы определяем стиль: шрифт красного цвета в 16 пунктов.

Форматирующий элемент <fo:block> содержит информацию обработки в виде элемента <xsl:process-children/>, который дает команду процессору о применении стилей и форматировании дочерних узлов элемента greeting, в данном случае текстового узла "Здравствуй, XSL!". Этот простой документ имеет только один дочерний узел, но если бы их было больше, обработка передавалась бы всем дочерним узлам (если только им не был бы присвоен другой стиль).

Потоковые объекты

Потоковый объект – это концепция, используемая при применении стилей. Когда документ печатается или просматривается, он «вытекает» на страницу или экран. Фактически каждый отдельный символ можно рассматривать как потоковый объект со своим собственным стилем. На практике, однако, рассматриваются более крупные потоковые объекты, такие как заголовок, абзац, фраза.

Обратите внимание на два термина, которые чаще всего будут встречаться в этой главе. Это – *встроенный потоковый объект* и *блочный потоковый объект*. Под встроенным потоковым объектом, в основном, понимается потоковый объект, до и после которого нет перевода строки, а до и после блочного потокового объекта такой перевод имеется.

В HTML элементы SPAN, I, B, EM являются примерами встроенных потоковых объектов, а элементы P, DIV, Hn и PRE – примеры блочных потоковых объектов.

Что представляют собой шаблоны XSL

В простейшем варианте таблица стилей XSL – это XML-документ, содержащий правила шаблона.

Как и любой XML-документ, таблица стилей должна соответствовать требованиям XML, предъявляемым к правильному документу. По сути дела, это означает, что должен существовать уникальный открывающий и закрывающий тэг (на практике данному условию удовлетворяет тэг <xsl:stylesheet></xsl:stylesheet>), а все другие тэги должны быть вложены.

Основной строительный блок таблиц стилей XSL – это правило шаблона, которое описывает, как узел-элемент первоначального XML-документа преобразуется в узел-элемент XSL, который в конце концов должен быть отформатирован и показан. Как уже говорилось, правило шаблона состоит из образца и действия.

В соответствии с образцом из исходного документа выбирается узел (обычно элемент) на основании следующих критериев:

- имя;
- происхождение элемента;
- идентификатор ID;
- универсальный шаблон (wildcard);
- атрибуты исходного элемента;
- положение элемента по отношению к его братьям;
- уникальность элемента по отношению к его братьям.

Часть *действие* шаблона включает в себя:

- элемент, создающий форматирующий объект в результирующем дереве. Элемент `<fo:block></fo:block>`, например, создает форматирующий блочный объект (см. ниже примечание 1);
- стиль, который нужно применить к форматирующему объекту, представленный в форме атрибутов. Например, `<fo:block font-size="16pt">` указывает, что к форматирующему объекту следует применить стиль в виде шрифта размером 16 пунктов (см. ниже примечание 2);
- тип обработки, которую нужно осуществить; он обычно бывает представлен в виде пустого тэга. Например, запись тэга `<xsl:process-children/>` приведет к тому, что процессор будет обрабатывать все дочерние узлы соответствующего элемента с помощью стиля и форматирования, описанных в других частях правила (см. ниже примечание 3 и 4).

Совет

Примечания:

1. *В таблицах стилей XSL «старого» типа можно было использовать как форматирующие объекты языка DSSSL, так и форматирующие объекты языка HTML. Для таблиц стилей XSL «нового» типа спецификация предоставляет список потоковых объектов, определяемых пространством имен fo:. Они подробно описаны ниже в разделах «Форматирующие объекты, задающие размещение» и «Потоковые объекты для содержания».*
2. *В большинстве случаев свойства стилей копируют свойства CSS, за исключением синтаксиса, который теперь выглядит следующим образом: [свойство стиля]=["значение"].*
3. *Рабочая группа ясно дает понять, что имя process-children может измениться.*
4. *Очень вероятно, хотя еще и не утверждено, что в «новом» XSL, так же как и в CSS, потомки потокового объекта будут наследовать многие свойства стилей.*

Обрабатывающая часть может также выполнять следующие операции:

- генерировать новые объекты стилей;
- добавлять текст, состоящий из букв;
- дублировать и размножать потоковые объекты;
- фильтровать элементы;

- изменять порядок следования элементов (выполнение этого пункта ожидается в будущих версиях XSL).

Построение дерева XSL

Процессор XML/XSL создает из исходного XML-документа дерево, называемое исходным деревом. Затем он использует таблицу стилей XSL для построения из него результирующего дерева. Во время этого этапа процессор может изменить порядок или продублировать узлы исходного дерева (или то и другое вместе), а также добавить новые потоковые объекты в форме элементов или текста.

Совет *Если вы все еще путаетесь с терминами «дерево», «узел», «родительский», «дочерний» и т.п., обратитесь к шестой главе, посвященной объектной модели документа XML.*

К узлам уже готового результирующего дерева применяются стили для создания оформленных стилями потоковых объектов.

Обратите внимание, насколько этот подход отличается от концепции типа CSS, где стили применяются непосредственно к исходному дереву.

Наличие дополнительного дерева в XSL означает, что окончательная запись после применения таблиц стилей XSL может иметь совершенно другую структуру по сравнению со структурой исходного дерева. Это расширяет диапазон возможностей настройки итогового отображения.

Различие между процессами применения стилей в XSL и CSS иллюстрруют схемы, представленные на рис. 8.1 и 8.2.

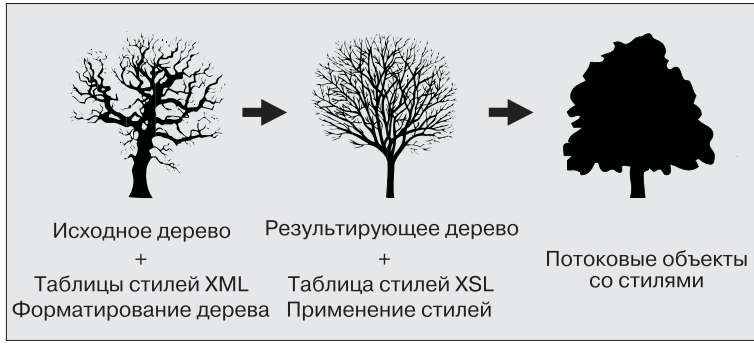


Рис. 8.1. Схема применения стилей в языке XSL

Возможность «перерисовывать» исходное дерево не только расширяет функциональные возможности, но и существенно обогащает применение стилей, возможное только в XSL. В то же время, если бы мы захотели добавить потоковый объект при использовании таблиц стилей CSS, нм пришлось бы изменять исходный документ.

Исходный документ

Вот пример короткого XML-документа `dolly.xml` о всемирно известной клонированной овце, отвечающего критерию правильности (опять же не обращайте

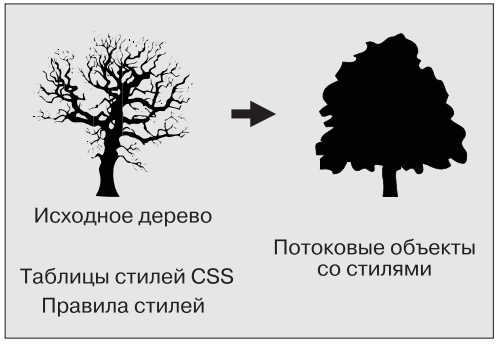


Рис. 8.2. Схема применения стилей в CSS

внимание на синтаксис, позже все будет объяснено):

```
<xdoc>
<greeting>Hello Dolly!</greeting>
<question>Can you clone me?</question>
<sheepobject>Dolly</sheepobject>
</xdoc>
```

Чтобы просмотреть этот документ, поставим ему в соответствие как таблицу стилей типа CSS, так и таблицу стилей типа XSL. Начнем с таблицы стилей CSS.

Использование таблицы стилей CSS

Ниже вы видите таблицу стилей типа CSS, `dolly.css`:

```
xdoc{
    display:block;
    font-family:times new roman,serif;
    font-size:12pt;
}
question{
    display:block;
}
sheepobject{
    display:block;
    font-size:16pt;
}
```

При использовании такой таблицы стилей дерево документа (рис. 8.3) остается неизмененным.

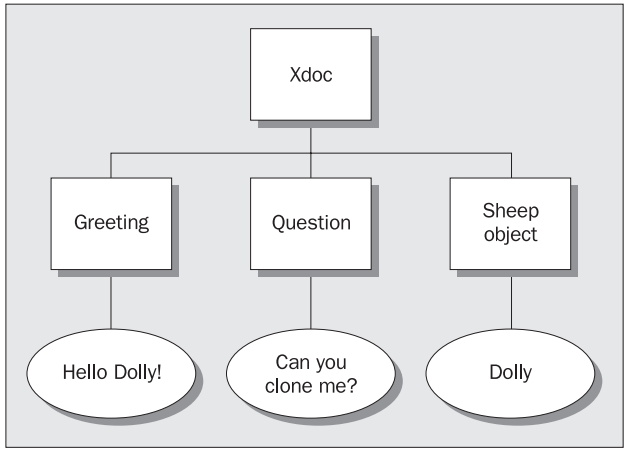


Рис. 8.3. Дерево документа при тображении с помощью каскадной таблицы стилей

При просмотре в браузере, совместимом с таблицами стилей CSS, докумнт будет выглядеть так, как показано на рис. 8.4.

Рис. 8.4. С каскадной таблицей стилей Долли не клонируется

Использование таблиц стилей XSL

Теперь применим к тому же документу таблицу стилей языка XSL. В следующем коде мы не только добавили некоторый текст в результирующее дерево с помощью элемента `<xsl:text>`, но и четыре раза обработали дочерний текст элемента `<sheepobject>` .

Не беспокойтесь, если вы не вполне поняли код, его детальным разбором мы займемся в следующем разделе. Просто запомните, что мы обработали один и тот же дочерний объект четыре раза, а также добавили некоторый текст (в терминологии объектной модели документа создали текстовый узел). Вот таблица стилей типа XSL:

```
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/TR/WD-xsl"
    xmlns:fo="http://www.w3.org/TR/WD-xsl/FO"
    result-ns="fo">

    <xsl:template match="/">
        <fo:page-sequence
            font-family="times new roman,serif"
            font-size="12pt">
            <xsl:apply-templates/>
        </fo:page-sequence>
    </xsl:template>

    <xsl:template match="*">
        <fo:block
            font-family="times new roman,serif"
            font-size="12pt">
            <xsl:apply-templates/>
        </fo:block>
    </xsl:template>

    <xsl:template match="sheepobject">
        <fo:block
            font-size="16pt">
            <xsl:text>YES I CAN!</xsl:text>
            <xsl:apply-templates/>
            <xsl:apply-templates/>
            <xsl:apply-templates/>
            <xsl:apply-templates/>
        </fo:block>
```

```
    </xsl:template>
</xsl:stylesheet>
```

Документ XSL использует исходное дерево для создания результрующего дерева, вид которого представлен на рис. 8.5.

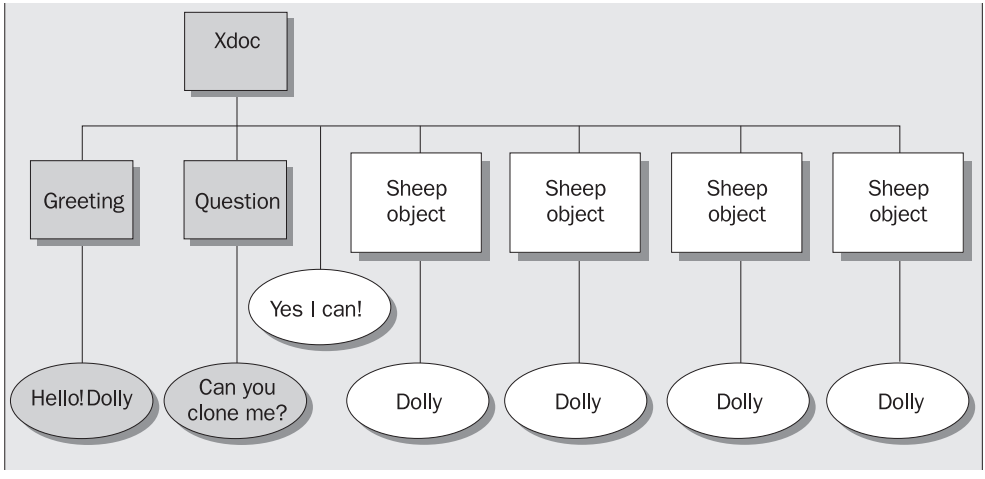


Рис. 8.5. Дерево документа при отображении с помощью языка XSL. Для ясности узлы исходного документа выделены серым цветом

В браузере, совместимом с языками XML и XSL, результат будет выглядеть примерно так, как показано на рис. 8.6.

Рис. 8.6. С помощью языка XSL Долли клонируется без труда

Совет *Заметим, что изображения на экранах в этом и предыдущем разделах получены искусственно. Мы вручную преобразовали XML файл в HTML чтобы показать, как он будет выглядеть в браузере, совместимом с XML и таблицами стилей.*

Построение результирующего дерева из исходного

Рассмотрим вкратце, как XSL-процессор выстраивает результирующее дерево из исходного. Для этой цели он использует *рекурсию*.

Рекурсия – это процесс, при выполнении которого программа повторно вызывает саму себя. Посмотрим, как это происходит в примере, разобранном выше. Вспомним еще раз пресловутую Долли:

```
<xdoc>
<greeting>Hello Dolly!</greeting>
<question>Can you clone me?</question>
<sheepobject>Dolly</sheepobject>
</xdoc>
```

Процессор переходит к исходному документу и начинает с элемента `xdoc`. На этом этапе он замечает, что данному элементу поставлены в соответствие определенные правила стилей XSL, а также указание обработать его дочерние объекты. Прежде чем применить стили, процессор просматривает дочерние объекты. Элемент `xdoc` имеет три таких объекта, и процессор сначала переходит к первому из них, `greeting`, находит правило стиля, затем определяет, что ему нужно обработать еще и дочерние объекты элемента `greeting`, он это и делает. Процессор обнаруживает, что текстовый узел, конечно же, не имеет дочерних объектов, а потому возвращается и начинает обрабатывать второй дочерний элемент, `question`. Хороший пример кода, использующего рекурсию, вы можете найти в главе 6 в разделе «Некоторые простые реализации».

Рассмотрим описанный процесс более детально. Вновь обратимся к нашей таблице стилей:

```
<xsl:stylesheet
   xmlns:xsl="http//www.w3.org/TR/WD-xsl"
   xmlns:fo="http//www.w3.org/TR/WD-xsl/FO"
   result-ns="fo">

   <xsl:template match="/">
       <fo:page-sequence
           font-family="times new roman, serif"
           font-size="12pt">
           <xsl:apply-templates/>
       </fo:page-sequence>
   </xsl:template>

   <xsl:template match="*">
       <fo:block
           font-family="times new roman,serif"
           font-size="12pt">
           <xsl:apply-templates/>
       </fo:block>
   </xsl:template>

   <xsl:template match="sheepobject">
       <fo:block
           font-size="16pt">
           <xsl:text>YES I CAN!</xsl:text>
           <xsl:apply-templates/>
           <xsl:apply-templates/>
           <xsl:apply-templates/>
           <xsl:apply-templates/>
       </fo:block>
   </xsl:template>
</xsl:stylesheet>
```

Корневому узлу <xdoc> соответствует образец в правиле шаблона:

```
<xsl:template match="/">
```

где символ "/" представляет собой сокращенное обозначение корня – в данном случае корневого элемента исходного документа. (Скоро мы рассмотрим и синтаксис.)

Когда процессор находит образец, он ищет инструкции, сообщающие, что делать дальше. Первая инструкция, которую он находит, касается создания потокового объекта в виде *последовательности страниц* (page sequence):

```
<fo:page-sequence ...
```

(Позже мы объясним, что такое потоковый объект типа page sequence.) Кроме того, процессор находит в виде атрибута начального тэга fo:page-sequence инструкцию, как выводить текст:

```
font-family="times new roman,serif"
font-size="12pt">
```

Таким образом, текст будет выведен шрифтом Times New Roman или, если его на машине не окажется, применяемым по умолчанию шрифтом 12 пунктов с засечками.

По мере продвижения по правилу стиля процессор находит инструкцию:

```
<xsl:apply-templates/>
```

сообщающую о необходимости обработки всех дочерних узлов элемента xdoc, что он и делает, начиная с поиска образца для каждого из дочерних узлов.

Первым дочерним элементом, с которым сталкивается процессор, является элемент greeting. Процессор не находит образца для элемента greeting и поэтому использует универсальный образец "*":

```
<xsl:template match="*">
```

который приводит к созданию блочного потокового объекта fo:block:

```
<fo:block
    font-family="times new roman,serif"
    font-size="12pt">
    <xsl:apply-templates/>
</fo:block>
```

В результате процессор применяет к элементу greeting и его дочерним узлам стиль со шрифтом Times New Roman размером 12 пунктов.

Затем процессор переходит к следующему дочернему узлу элемента xdoc, элементу question и, не найдя соответствующего правила шаблона, применяет универсальный образец во второй раз.

Закончив с элементом question, процессор переходит к элементу sheepobject и находит соответствующий ему образец:

```
<xsl:template match="sheepobject">
```

Затем он создает потоковый объект fo:block, который содержит атрибут, указывающий, что потоковый объект должен быть отформатирован как текст шрифтом 16 пунктов:

```
<fo:block
   font-size="16pt">
```

Внутри блока первой инструкцией является:

```
<xsl:text>YES I CAN!</xsl:text>
```

которая создает текстовый потоковый объект со значением строки, равным `"YES I CAN!"` и вставляет его в результирующее дерево как первый дочерний узел блочного потокового объекта.

Следующая инструкция, находящаяся внутри блока

```
<xsl:apply-templates/>
```

создает текстовый потоковый объект (размер шрифта 16 пунктов) из строки `"Долли"`. Существуют еще три инструкции `'xsl:apply templates'`, которые нужно выполнить:

```
<xsl:apply-templates/>
<xsl:apply-templates/>
<xsl:apply-templates/>
```

Каждая из этих инструкций создает из строки `"Долли"` новый текстовый потоковый объект и помещает его в результирующее дерево. После этой операции результирующее дерево содержит уже четыре копии потокового объекта `"Долли"`.

Данный процесс представляет собой один из видов рекурсии. XSL-процессор обходит дерево, создавая «результирующие» потоковые объекты в соответствии с указанной информацией.

Наследование

Заметим, что хотя обрабатываются только дочерние узлы данного элемента, стиль, применяемый к конкретному дочернему узлу, будет распространяться на всех его потомков, пока не последует иное предписание. Другими словами, если для корневого элемента указан текст размером 16 пунктов, текст *всех* элементов-потомков тоже будет 16 пунктов до тех пор, пока для того или иного элемента-потомка не поступит какая-либо другая команда. Тогда новый стиль таким же образом будет распространяться на всех потомков этого элемента.

Сопоставление по умолчанию

Если хотя бы в одном из правил шаблонов инструкция `process-children` была опущена, то ни один из потомков данного узла не будет обработан. (Это отличный способ намеренно не показывать часть материала, включенного в исходный документ; о данном приеме мы расскажем позже.)

А что случится, если мы просто-напросто не напишем правило шаблона? Это приведет к остановке процесса рекурсии, поэтому для предотвращения подобной ситуации существует особое встроенное правило. Оно применяется и к узлам элементов, и к корневому узлу.

Эквивалентом встроенного правила шаблона является следующий код:

```
<xsl:template match="*|/">
   <xsl:apply-templates/>
</xsl:template>
```

Встроенное правило предотвращает остановку обработки, поскольку образец создан для корневого элемента `"/"` ИЛИ (символ `'|'`) любых дочерних узлов (`*`), для которых не найден конкретный шаблон.

Перед детальным обсуждением правил шаблона рассмотрим, как в языке XSL для определения и идентификации элементов используются пространства имен.

Пространства имен и таблицы стилей XSL

Спецификация XSL в полном объеме использует пространства имен. Знакомство с этой методикой, несомненно, должно упростить освоение материала, связанного с их применением в таблицах стилей XSL, однако глубокое знание в этой области здесь совсем не обязательно. Мы дадим объяснение, хотя и очень упрощенное, но достаточное для понимания того, какую роль играют эти пространства в различных составляющих элемента таблицы стилей.

Если же вы относитесь к тем пользователям, кому нравится начинать с азов, обратитесь к спецификации на сайте http://www.w3.org/TR/WD-xml-names.

Все таблицы стилей, соответствующие спецификации XSL, должны содержаться в корневом элементе:

```
<xsl:stylesheet>
...
</xsl:stylesheet>
```

Это уже было показано ранее, в нашем втором примере таблицы стилей. Однако еще раз приведем весь элемент (обратите внимание, что у него есть три атрибута: `xmlns:xsl`, `xmlns:fo` и `result-ns`).

```
<xsl:stylesheet
   xmlns:xsl="http//www.w3.org/TR/WD-xsl"
   xmlns:fo="http//www.w3.org/TR/WD-xsl/FO"
   result-ns="fo">
...
</xsl:stylesheet>
```

Все таблицы стилей, соответствующие спецификации XSL, должны иметь первый атрибут и его значение точно в таком виде, как указано в официальном определении `xmlns:xsl="http//www.w3.org/TR/WD-xsl"` .

Совет

1. *На самом деле официальное определение типа XML-документа для XSL показывает значение атрибута xmlns:xsl как фиксированное (#FIXED), поэтому если даже вы его не написали, его присутствие подразумевается, при условии, что вы ссылаетесь на определение типа документа (так же, как вы сослались бы для обычного XML-документа, используя идентификатор SYSTEM "... .dtd" в декларации <!DOCTYPE... в самом начале таблицы стилей).*
2. *Сочетание xmlns представляет собой зарезервированное ключевое слово языка XML. За ним всегда следует двоеточие и аббревиатура (например, xsl или fo), затем знак равенства и адрес URI. Таким образом, если процессор встречается с элементом, префикс которого xsl: или fo:, он «знает», что этот элемент принадлежит пространству имен http//www.w3.org/TR/WD-xsl или пространству имен http//www.w3.org/TR/WD-xsl/FO.*

Смысл этого условия состоит в передаче процессору информации о том, что когда бы и где бы он ни встретил элемент с префиксом `xsl:`, он должен интерпретировать его в соответствии со спецификацией XSL. Другими словами, все подобные элементы находятся в пространстве имен `xsl`.

Второй атрибут теоретически необязателен, но, исходя из практических соображений, его необходимо включать. Он сообщает процессору, что элементы с префиксом `fo:` (сокращение для потоковых объектов – flow objects) находятся в пространстве имен `fo`, и стили, которые будут применяться, должны использовать словарь форматирования, описанный в спецификации XSL. Таким образом, когда процессор распознает элемент `<fo:block font-size="10pt">`, он будет знать, что таблица стилей ссылается на синтаксис, находящийся в пространстве имен `fo`.

Совет *Конечно, можно использовать другой тип словаря стилей, например тот, который можно найти на сайте http://www.mystyle.org, и если вы хотите потратить силы на создание такого словаря, а также убедить производителей приложений его поддержать, желаем вам удачи!*

Третий атрибут – `result-ns="fo"` – просто сообщает процессору, что создается результирующее дерево, использующее словарь форматирования `fo`.

Пространства имен для стилей

В настоящее время существует предложение использовать в качестве пространства имен словарь каскадных таблиц стилей CSS. Мы обсудим это предложение позже, в разделе «Пространство имен таблиц стилей CSS». Сообщение консорциума W3C можно найти по адресу

http//www.w3.org/TR/NOTE-XSL-and-CSS

Атрибут этого пространства имен должен быть описан в корневом элементе `xsl:stylesheet` следующим образом:

```
xmlns:css="http//www.w3.org/TR/NOTE-XSL-and-CSS"
```

При этом процессор должен быть информирован о том, что мы строим дерево, используя пространство имен `css`. Эта операция выполняется записью:

```
result-ns="css"
```

Атрибуты элемента xsl:stylesheet

Атрибуты, которые может иметь элемент `xsl:stylesheet`, показаны ниже. Большинство из них описаны в официальном определении типа документа и вполне понятны без развернутых разъяснений:

```
<!ATTLIST xsl:stylesheet
   result-ns NMTOKEN #IMPLIED
   default-space (preserve|strip) "preserve"
   indent-result (yes|no) "no"
```

```
    id ID #IMPLIED
    xmlns:xsl CDATA #FIXED "http://www.w3.org/TR/WD-xsl"
>
```

Атрибут result-ns

Как уже было сказано, этот атрибут информирует процессор о том, что будет построено дерево. Обычно дерево формируется с использованием пространства имен `fo` (если требуется, чтобы документ соответствовал спецификации XSL и мог быть интерпретирован в ее терминах).

Атрибут default-space

Известно, что обычной практикой в языке HTML является удаление или, скорее, «схлопывание» пустого пространства. В языке XML (и XSL) пустое пространство, как правило, сохраняется.

Атрибут indent-result

Только об этом атрибуте можно сказать, что его предназначение изложено в спецификации не совсем понятно. Возможно, он указывает, что весь текст на экране должен быть сдвинут на определенное расстояние вправо.

Теперь самое время перейти к обсуждению шаблонов и синтаксиса выражений, которые могут быть использованы для указания, какой части XML-документа должен соответствовать шаблон стиля.

Правила шаблона таблиц стилей XSL

Основой таблиц стилей XSL является правило шаблона, которое создает трафарет, позволяющий агенту пользователя создавать результирующий узел с примененным к нему стилем из исходного узла. Мы уже видели тэг `<xsl:template>` в действии.

Как мы говорили ранее, шаблон имеет две части: *образец* для сопоставления и обрабатывающее *действие*.

Часть сопоставление (matching part) указывает исходный узел XML, который нужно обработать. Информация для сопоставления содержится в атрибуте `match`.

Обрабатывающая часть (processing part) определяет, каким операциям должны быть подвергнуты дочерние узлы и какие стили к ним нужно применить. Эта информация (элементы, определяющие потоковые объекты и обработку) содержится в дочерних элементах шаблона.

В следующих разделах мы обсудим различные методы установления соответствия (сопоставления). После чего более детально обсудим форматирование потоковых объектов.

Простые сопоставления

Здесь мы рассмотрим простые сопоставления по образцу, включающие сопоставления по имени, происхождению, потомкам, положению и идентификатору ID.

Сопоставление по имени

При таком сопоставлении исходный элемент просто идентифицируется по своему имени с помощью атрибута `match`, как в нашем примере, приведенном выше. Значение, которое мы присваиваем атрибуту `match`, называется *образцом*.

Таким образом, строка

```
<xsl:template match="greeting">
```

ставит в соответствие данный шаблон всем элементам `<greeting>` исходного документа.

Сопоставление по происхождению

Также как и в таблицах стилей CSS, можно осуществить сопоставление по происхождению элемента.

Согласно правилу CSS описание стиля

```
P EM {[вставьте здесь описания стиля]}
```

будет соответствовать любому элементу `EM`, который имеет предка `P`.

Чтобы создать аналогичный образец для сопоставления в языке XSL, необходимо записать правило следующим образом:

```
<xsl:template match="P//EM">
```

Обратите внимание на двойную косую черту – это обязательный синтаксис. Двойная косая черта указывает на то, что элемент `EM` может быть на один или большее количество уровней вложенности ниже, чем его предок `P`. Таким образом, отметка этим знаком приведет к выбору того элемента `EM`, который где-то среди своих предков имеет элемент `P`.

Например, будет установлено соответствие со следующим элементом `<EM>`, расположенным ниже:

```
<P>
   <FIRST>
   <SECOND>
       ...
           ...
               <EM>
                   Текст, к которому нужно применить стиль
               </EM>
           ...
       ...
   </SECOND>
   </FIRST>
</P>
```

Для сопоставления с элементом `EM`, который является дочерним элементом элемента `P`, запишем:

```
<xsl:template match="P/EM">
```

Заметим, что здесь мы использовали одинарную косую черту, указывающую на строгое соотношение «родительский элемент – дочерний элемент» без промежуточных уровней.

```
<xsl:template match="DIV/P/EM">
```

Эта запись установит соответствие с элементом `EM`, дед которого элемент `DIV`, а родитель – элемент `P`.

Сопоставление нескольких имен

В правиле стилей CSS можно объединить несколько селекторов вместе, например:

```
H1, H2, H3 {[вставьте здесь описание стиля]}
```

Эта запись установит соответствие с элементами `H1`, `H2`, `H3`.

Для получения такого же образца в правиле языка XSL мы должны отделить друг от друга исходные элементы с помощью вертикальной черты. Следующий шаблон

```
<xsl:template match=" H1 | H2 | H3 ">
```

приведет к тому же результату, что и предыдущая запись для таблиц стилей CSS.

Сопоставление корня

Часто бывает полезным поместить все основные свойства стиля в корень документа, чтобы дать возможность всем дочерним узлам наследовать эти свойства.

Совет *Обратите внимание: корень – это не то же самое, что корневой элемент документа, с которым можно установить соответствие по имени; корень является корневым объектом всего документа, а в терминах объектной модели документа (см. главу 6) – узлом document. На самом деле он фиктивен. Такое различие проводится потому, что если бы узел document и корневой элемент документа совпадали, и мы бы указали корневой элемент в нашем шаблоне стилей, например <xsl:template match="xdoc">, тогда при использовании <xsl:process-children/> корневой элемент не был бы обработан.*

Правильный результат достигается путем использования одиночной косой черты, которая представляет узел `document`. Сопоставление по образцу

```
<xsl:template match="/">
```

приводит к выбору корневого узла.

Сопоставления по универсальному образцу

Для создания образца, которому можно сопоставить любые элементы XML-документа с незаданным шаблоном, необходимо использовать универсальный селектор `"*"`. Следующий тэг шаблона

```
<xsl:template match="*">
```

будет соответствовать каждому узлу в исходном документе (если для какого-то конкретного узла не определено другое правило). Обратите внимание, что

шаблон

```
<xsl:template match="/|*">
```

будет соответствовать всему документу, поскольку косая черта соответствует корню, а звездочка – всем элементам. Это может быть полезно для определения правила, которое нужно применить, если других правил не указано. Агент пользователя, поддерживающий XSL, обязан иметь описанное ниже встроенное правило.

Встроенное правило шаблона

Как мы уже видели, исходное дерево документа ставится в соответствие результирующему выходному дереву с помощью процесса рекурсии. Чтобы предотвратить остановку процесса рекурсии вследствие отсутствия подходящего правила, совместимый с языком XSL агент пользователя обязан подставлять встроенное правило шаблона, если он не находит никакого другого (как мы упоминали ранее в разделе «Сопоставление по умолчанию»). Напоминаем, что встроенное правило имеет следующий вид:

```
<xsl:template match="*|/">
   <xsl:process-children/>
</xsl:template>
```

Но ничто не мешает нам написать собственное правило. Например, код

```
<xsl:template match="*|/">
[не общедоступный материал]
</xsl:template>
```

приводил бы к остановке обработки всех узлов, для которых не указано правила стиля, и подстановке строки `"[не общедоступный материал]"` в результирующее дерево.

Сопоставление по идентификатору ID

Синтаксис для сопоставления по идентификатору ID такой же как синтаксис указателей XPointer (см. главу 5), а именно:

```
<xsl:template match="id([id находится здесь])">
```

Таким образом, в XML-документе может быть написано:

```
<greeting id="A1">Здравствуй, мир!</greeting>
```

Следующая строка в таблице стилей XSL была бы сопоставлена элементу, атрибут ID которого имеет значение `"A1"`. В данном случае таким элементом является `<greeting...>`.

```
<xsl:template match="id(A1)">
```

Сопоставление по атрибуту

Синтаксис для сопоставления по атрибуту следующий:

```
<xsl:template match="имя элемента [attribute (имя атрибута)=значение атрибута]">
```

Совет *Тот факт, что спецификация использует квадратные скобки для имени и значения атрибута, может внести некоторую путаницу. Как вам, вероятно, известно, квадратные скобки традиционно используются для заключения в них обобщенного термина при определении кода.*

Для того чтобы увидеть, как работает сопоставление по атрибуту, рассмотрим следующий XML-документ, к которому мы хотим применить стиль:

```
<xdoc>

   <chapter num="ch1">
       <chtitle>первый</chtitle>
       <para type="opening">Некоторый текст</para>
       <para type="regular">Некоторый <emph>выделенный</emph> текст</para>
       <para type="closing">Еще немного текста</para>
   </chapter>

   <chapter num="ch2">
       <chtitle>второй</chtitle>
       <para type="opening"> Некоторый текст </para>
       <para type="regular">Некоторый <emph>выделенный</emph>текст</para>
       <para type="closing"> Еще немного текста </para>
   </chapter>

   <chapter num="ch3">
       <chtitle>третий</chtitle>
       <para type="opening"> Некоторый текст </para>
       <para type="regular">Некоторый <emph>выделенный</emph>текст</para>
       <para type="closing"> Еще немного текста </para>
   </chapter>
</xdoc>
```

В нашей таблице стилей XSL можно использовать следующие шаблоны сопоставления:

```
<xsl:template match="para">
```

для сопоставления со всеми элементами `para`;

```
<xsl:template match="para[attribute(type)]">
```

для сопоставления со всеми элементами `para` с атрибутом `type`; и

```
<xsl:template match="para[attribute(type)='opening']">
```

для сопоставления со всеми элементами `para` с атрибутом `type` и значением `'opening'`.

Можно также использовать шаблон

```
<xsl:template match="chapter[attribute(num)='ch1']/para[attribute(type)='opening'
]">
```

для сопоставления со всеми элементами `para` с атрибутом `type` и значением `'opening'`, которые имеют родителя `chapter` с атрибутом `num` и значением `'ch1'` (то есть самый первый элемент `para` в нашем примере).

Сопоставление по дочернему узлу

Элементы могут быть сопоставлены по спецификатору дочернего узла.

Общий вид синтаксиса, указанный в спецификации, выглядит следующим образом:

```
<xsl:template match="имя элемента [имя дочернего узла]">
```

Снова обращаем ваше внимание на необычное употребление квадратных скобок.

В примере, приведенном выше:

```
<xsl:template match="para[emph]">
```

будет сопоставлено всем элементам `para` с дочерним элементом `emph`.

Сопоставление по положению

Язык XSL позволяет делать сопоставления по образцу, использующему спецификаторы положения.

В табл. 8.1 перечислены различные спецификаторы положения, определенные в спецификации.

Синтаксис для сопоставления по спецификаторам положения следующий:

```
<xsl:template match="имя элемента [описание положения]">
```

Опять-таки обращаем ваше внимание на необычное употребление квадратных скобок.

Таблица 8.1. Спецификаторы положения

first-of-type()	Элемент должен быть первым братом, имеющим данный тип
not-first-of-type()	Элемент не должен быть первым братом, имеющим данный тип
last-of-type()	Элемент должен быть последним братом, имеющим данный тип
not-last-of-type()	Элемент не должен быть последним братом, имеющим данный тип
first-of-any()	Элемент должен быть первым братом среди элементов любого типа
not-first-of-any()	Элемент не должен быть первым братом среди элементов любого типа
last-of-any()	Элемент должен быть последним братом среди элементов любого типа
not-last-of-any()	Элемент не должен быть последним братом среди элементов любого типа
only-of-type()	Элемент не должен иметь элементов-братьев того же типа
not-only-of-type()	Элемент должен иметь один или более элементов-братьев того же типа
only-of-any()	Элемент вообще не должен иметь элементов-братьев
not-only-of-any()	Элемент должен иметь один или более элементов-братьев

Для нашего примера XML-документа шаблон для сопоставления можно определить как

```
<xsl:template match="chapter [first-of-type()]">
```

который будет соответствовать элементу `chapter` со значением атрибута `id` равным `ch1`. Еще можно использовать шаблон

```
<xsl:template match="chapter [last-of-type()]">
```

для сопоставления с элементом `chapter`, в котором значение `id` равно `ch3`.

Разрешение конфликтов сопоставлений

Может оказаться так, что к одному и тому же исходному элементу применяется более, чем одно правило шаблона. Правила в таком случае работают следующим образом:

- образец для сопоставления, который ссылается на указанный атрибут ID элемента, считается более конкретным, чем тот образец, который не ссылается на ID;
- если оба образца ссылаются на атрибут ID элемента или если никакой из образцов не ссылается, тогда более конкретным является образец с большим количеством компонентов. (Под компонентами мы понимаем имена элементов или атрибутов, а также любые определенные значения атрибутов.)

В табл. 8.2 приведены примеры правил из спецификации (раздел 2.6.11), выстроенные в порядке убывания конкретности.

Таблица 8.2. Правила спецификации в порядке убывания конкретности

1	id(employee-of-the-month)
2	employee[attribute(type)='contract', attribute(country)='USA']
3	employee[attribute(type)='contract']
4	employee
5	*

Заметим, что шаблон с идентификатором `id` всегда наиболее конкретен, а наименее конкретен универсальный шаблон `"*"`.

Форматирующие объекты, задающие размещение

В этом разделе мы рассмотрим форматирующие объекты, задающие размещение на странице. Они определены в последней версии спецификации XSL.

Простые форматирующие объекты

«Старый» XSL использовал группу форматирующих объектов DSSSL и HTML. «Новый» XSL предоставляет свой собственный список форматирующих объектов, которые определены в спецификации и на которые можно ссылаться с использованием пространства имен `fo:`. Форматирующий объект применяется к узлу результирующего дерева, поскольку он содержится в части *действие* шаблона.

Общий вид синтаксиса следующий:

```
<xsl:template match="[pattern]">      <!-- Определяет образец. -->
<fo:[ форматирующий объект] ([свойство стиля]="[значение]")*>
       [инструкции обработки]*
   </fo:[ форматирующий объект]>
</xsl:template>
```

Несмотря на то, что в будущем должно появиться несколько стилей размещения, спецификация от августа 1998 года определяет лишь *объект-последовательность страниц* (page-sequence) и *объект-распорядитель простой страницы* (simple-page-master).

Объект-последовательность страниц действует как предок ряда страниц, предназначенных либо для распечатывания, либо для демонстрации в Internet. Спецификация обещает, что в будущем для других типов форматирования будет введен в действие *объект-схема потока* (flow-map).

Объект page-sequence

Объект-последовательность страниц действует как объект-предок, который содержит от одного до нескольких потоковых объектов или последовательностей страниц. Он может содержать один или более дочерних объектов `simple-page-master` или до шести дочерних объектов `queues` (см. ниже).

На практике он работает как средство, содержащее правила стилей, которые могут наследоваться остальным документом. Например, в правиле шаблона

```
<xsl:template match="/">
    <fo:page-sequence
        font-family="times new roman,serif"
        font-size="12pt"
        background-color= "white'
        color= "black">
        <xsl:process-children/>
    </fo:page-sequence>
</xsl:template>
```

правила `font` и `color` могут наследоваться другими объектами потока в результирующем дереве, если не будут указаны альтернативные правила. Это применение объекта `page-sequence` аналогично указанию правил стилей в элементе `<BODY>` языка HTML.

Объект simple-page-master

Форматирующий объект `simple-page-master` описывается с помощью синтаксиса:

```
<fo:simple-page-master [свойства стиля]>
```

Этот потоковый объект описывает простую страницу, которая может быть разделена на шесть областей. (Схема приведена в разделе, посвященном объекту `queue`.) Такая модель может быть использована или для печати, или для демонстрации страниц в Internet.

Элемент `fo:simple-page-master` *должен* иметь атрибут `master-name`, который может принимать одно из следующих значений:

- `first`: – форматирование как для первой страницы из серии страниц;
- `odd`: – эквивалент «левой страницы» в средах для печати;
- `Even`: – эквивалент «правой страницы» в средах для печати;
- `Scrolling`: – тип страницы, используемый в представлениях на экране.

Таким образом, код для этого форматирующего объекта будет выглядеть так:

```
<fo:simple-page-master master-name="first" [другие свойства]>
```

Редакторы спецификации обещают в будущих версиях более сложные объекты размещения, такие как, например, столбцы.

Теперь рассмотрим потоковые объекты более низкого уровня, которые будут находиться внутри страниц.

Потоковые объекты содержимого

Здесь приведены некоторые более традиционные форматирующие объекты содержания и их краткое описание.

Объект queue

Этот объект появляется в таблице стилей как элемент:

```
<fo:queue ...
```

Возможно, что объект *очередь* (`queue`) меньшего всего похож на потоковый. И действительно, он не относится к данному разряду в полном смысле этого слова, а лишь указывает место, куда следует поместить потоковый объект.

Объект `queue` может быть *только* дочерним к объекту `page-sequence`. Он имеет атрибут `queue-name`, который принимает одно из шести значений: `title` (заголовок), `header` (верхний колонтитул), `body` (основная часть), `footer` (нижний колонтитул), `start-side` (начальная сторона) и `end-side` (конечная сторона). Область `body` – это то место, куда попадает основное содержание документа.

На схеме (рис. 8.7) показаны эти шесть областей страницы.

Обратите внимание, что область `title` используется только для сред (устройств отображения) с прокруткой.

Из существующей на сегодняшний день спецификации совершенно непонятно, как потоковые объекты должны попадать в заданную область. Однако она ясно дается понять, что последующие редакции сообщат, каким образом использовать этот форматирующий объект для нумерации страниц, создания стиля словаря с помощью верхнего и нижнего колонтитулов и отображения записей на полях.

Рис. 8.7. Области на странице, определяемые значениями атрибута queue-name объекта queue

Объект sequence

Форматирующий объект *последовательность* (`sequence`) создается с помощью элемента:

```
<fo:sequence ...
```

Данный потоковый объект используется для объединения потоковых объектов, которые имеют унаследованную группу свойств. Несмотря на то, что `<fo:sequence ...` не является в точности тем же, что и *встроенный* (`inline`)

потоковый объект в таблице стилей CSS, он в основном служит тем же целям.

Рассмотрим следующий отрывок кода:

```
<xsl:template match= "emphasis">
    <fo:sequence font-style="italic">
        <xsl:process-children/>
    </fo:sequence>
</xsl:template>
```

В результирующем дереве он приведет к созданию такого потокового объекта, в котором ко всем элементам с именем `emphasis` будет применен курсив. В HTML как элемент `<SPAN>`, так и элемент `<EM>` являются примерами встроенных потоковых объектов.

Объект block

Ранее мы уже встречались с синтаксисом для этого элемента:

```
<fo:block...
```

Его использование ведет к созданию блочного потокового объекта. На практике это означает, что до него и после него имеется перевод строки. В HTML как элемент `<P>`, так и элемент `<DIV>` являются примерами блочных потоковых объектов.

Объект list

Этот потоковый объект создается с помощью элемента:

```
<fo:list...
```

Потоковый объект *список* (`list`) действует как контейнер для потоковых объектов `list-item` (пункт списка), `list-item-label` (метка пункта списка), `list-item-body` (содержание пункта списка). Схема (рис. 8.8) показывает, как они друг с другом связаны.

Заметим, что если требуется создать вложенные списки, второй список должен быть дочерним к потоковому объекту `list-item-body`.

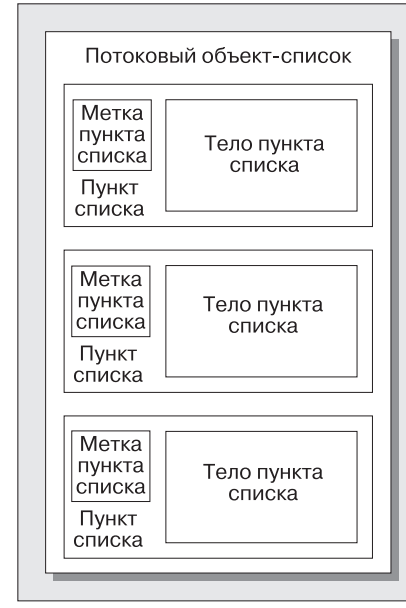


Рис. 8.8. Взаимосвязь составных частей потокового объекта-списка

Объект rule-graphic

Потоковый объект, называемый *линейка* (`rule-graphic`), появляется в виде:

```
<fo:rule-graphic...
```

Потоковый объект `rule-graphic` соответствует правилу `horizontal` (тэг `<HR>`) в языке HTML, но имеет дополнительную функцию. Он может быть как встроенным, так и блочным потоковым объектом.

Объект graphic

```
<fo:graphic...
```

Потоковый объект *графика* (`graphic`) соответствует потоковому объекту `<IMG>` в HTML, но опять-таки с дополнительной функцией. Он может быть как встроенным, так и блочным потоковым объектом.

Объект score

```
<fo:score...
```

Объект *черта* (`score`) представляет собой встроенный потоковый объект, с помощью которого можно подчеркивать или перечеркивать текст, накладывать сверху рамки и т.п.

Объект block-level-box

```
<fo:block-level-box...
```

Объект *рамка* (`box`) в XSL аналогичен рамкам в CSS. *Рамки* (boxes) в XSL используются для создания границ, полей, фонов и просвета между границей рамки и содержанием. Могут существовать *встроенные рамки* (`inline-box`) и *блочные рамки* (`block-level-box`).

Объект inline-box

```
<fo:inline-box ...
```

(См. описание объекта `block-level-box`.)

Объект page-number

```
<fo:page-number...
```

Потоковый объект *номер страницы* (`page-number`) используется для того, чтобы дать команду средству форматирования создать и вывести номера страниц в желаемом стиле.

Объект link

```
<fo:link...
```

Этот объект создает область, куда можно поместить потоковые объекты *локаторы места назначения* (`link-end-locator`). Потоковые объекты `link-end-locator` содержат информацию о месте назначения, на которое может сослаться пользователь. Если внутри объекта `link` (ссылка) существует более одного объекта `link-end-locator`, агент пользователя должен предоставить некоторое средство, например, выпадающий список, с помощью которого пользователь мог бы сделать выбор.

Объект link-end-locator

```
<fo:link-end-locator...
```

Потоковые объекты `link-end-locator` предоставляют информацию о месте назначения ссылки (эквивалентно `HREF="[URI]"` в HTML).

Объект character

Этот потоковый объект позволяет рассматривать единичный *символ* (`character`) в качестве потокового объекта. Он имеет следующий синтаксис:

```
<fo:character...
```

Теперь рассмотрим свойства стилей, которые могут быть указаны в виде атрибутов элементов, соответствующих объектам потока, описанным выше.

Применение стилей

Каждый форматирующий объект или потоковый объект может иметь определенные свойства стилей. Каждый форматирующий объект или потоковый объект может наследовать некоторые свойства стилей от своих узлов-предков. То, какие свойства могут быть унаследованы, зависит от конкретного объекта. При разработке языка XSL было приложено немало усилий, чтобы он оказался совместим с языком таблиц стилей CSS, и поэтому многие свойства стилей в обеих спецификациях имеют одни и те же названия. Однако в XSL существуют и некоторые дополнительные свойства (примером чего, кстати, выбранным наугад, является свойство `char-kern-mode`). Список всех возможных свойств стилей XSL в алфавитном порядке можно найти в разделе 3.24 спецификации XSL, опубликованной в августе 1998 года. Свойства стилей CSS перечисляются в «Приложении F» и «Приложении G».

Если вы знакомы с таблицами стилей CSS, то в следующем разделе сразу отыщете способ для создания простой таблицы стилей на языке XSL.

Преобразование CSS в XSL

Ниже приведено правило стиля CSS. Оно состоит из селектора и описания. Описание состоит из нескольких пар свойство-значение, разделенных точкой с запятой:

```
[selector]{font-name:times new roman, serif; font-size:14pt; background-
color:#800000; }
```

Чтобы преобразовать описание стиля CSS в описание стиля XSL, вставьте пары свойство-значение CSS в синтаксис XSL следующим образом:

```
<fo:[форматирующий объект] [название свойства CSS]="[значение CSS]">
```

так, чтобы приведенное выше описание преобразовалось в следующее:

```
<fo:[formatting object] font-name=" times new roman, serif" font-size="14pt"
   background-color="#800000">
```

Существует несколько свойств, уникальных для пространства имен `fo:`. В разделе «Более сложное применение стилей» мы продемонстрируем применение графического (`graphic`) потокового объекта, который использует достаточно большое число подобных свойств.

Простая обработка

Как уже было сказано, начальная часть процесса применения стилей состоит в описании потокового объекта, который должен быть создан, и свойств стилей, которые нужно применить. На следующем этапе включается обработка или построение потокового объекта результирующего дерева из исходного дерева.

Элемент xsl:process-children

Стандартная команда XSL-процессору выглядит следующим образом:

```
<xsl:process-children />
```

Она приводит к обработке всех дочерних узлов исходного элемента, с которым установлено соответствие.

В примере с клонированной овцой (файл `dolly.xml`) мы убедились, что при повторении данной команды получаются копии потоковых объектов.

Добавление текстового потокового объекта

Текстовый потоковый объект можно добавить простым приемом включения текста в модель обработки, поэтому в следующем коде:

```
<xsl:template match= "emphasis">
    <fo:sequence font-style="italic">
    Note Well!:-
    <xsl:apply-templates/>
    </fo:sequence>
</xsl:template>
```

текстовая строка `"Note Well!:-"` будет помещена перед каждым участком выделенного текста. Однако следует заметить, что при таком использовании пробельные литеры ликвидируются. Если необходимо их сохранить, текст следует поместить в элемент `xsl:text`.

Элемент xsl:text

Элемент `xsl:text` неявно содержит зарезервированный атрибут `xml:space` со значением по умолчанию `preserve`. В результате, внутри элемента `xsl:text` пробельные литеры сохраняются автоматически.

```
<xsl:text>
Можете ли вы сохранить пробельные литеры?

ДА, МОГУ!
</xsl:text>
```

Элемент xsl:process

Если требуется просто обработать какой-либо дочерний объект, нам поможет элемент `xsl:process`. Элементы, которые будут подвергнуты обработке, выбираются по имени.

Следующий код:

```
<xsl:template match= "book">
   <fo:block font-size= "12pt">
       <apply-templates= "chapter-title"/>
   </fo:block>
</xsl:template>
```

приведет к тому, что ко всем дочерним элементам `chapter-title` элемента `book` будет применен стиль со шрифтом в 12 пунктов.

Образцы для выбора (значения атрибута `select`) используются так же, как атрибуты `match` элемента `xsl:template`.

Утраченные форматирующие объекты

Из текущей спецификации выпали некоторые важные форматирующие объекты. Авторы спецификации обещают, что в ближайших версиях этот пробел будет восполнен. Вот некоторые из потоковых объектов, которые они обещают включить:

- таблицы;
- рамещение текста в виде нескольких столбцов и другие виды сложного расположение текста на странице;
- объекты, расположенные рядом друг к другу (сторона к стороне), и «плавающие» объекты;
- выдержки из содержания документа, например указатель, оглавление, заметки в конце страницы и на полях;
- международные объекты;
- добавочные строительные блоки, помимо нумерации страниц;
- математику.

Совет *Заметим также, что текущая спецификация XSL не поддерживает правило, аналогичное правилу @media таблиц стилей CSS, которое позволяет определить специальные стили для конкретных методов вывода, таких как, например, экран, обычная печать, устройство для чтения незрячими.*

Сложное применение стилей

Большинство свойств стилей, которые мы рассмотрели, аналогичны свойствам стилей CSS. Спецификация XSL предоставляет и некоторые уникальные свойства. Там же можно найти полный список свойств стилей. Однако следует иметь в виду, что в будущих версиях рабочего проекта этот перечень может быть дополнен или изменен.

Поскольку совместимого с языком XSL браузера не существует (кроме, возможно, вскоре выходящей пробной версии Microsoft Internet Explorer 5) этот список предлагается просто как иллюстрация богатства возможностей форматирования, которые будут доступны после внедрения XSL.

Ниже в качестве примера приведен список описанных в перечне свойств потокового объекта `graphic`. Обратите внимание, что некоторые из них имеют

комментарии в скобках, начинающиеся или с аббревиатуры DSSSL:, или с аббревиатуры CSS:. Они указывают, является ли имя свойства таким же (*-то же самое-*) в языке DSSSL, а также в CSS, или данное свойство имеет другое имя, характерное для той или иной спецификации. Термин *не основное* означает, что устройство форматирования XSL не обязано реализовывать этот объект или свойство, но должно включать действие по умолчанию для предотвращения ошибок обработки, которые могут возникнуть там, где таблица стилей указывает эти свойства.

Итак, потоковый объект `graphic` в языке XSL имеет следующие свойства:

- inline (*встроенный*);
- block-level-alignment (*выравнивание по уровню блока*);
- break-after (*разрыв после*) (DSSSL: *-то же самое-*);
- break before (*разрыв до*) (DSSSL: *-то же самое-*);
- color (*цвет*) (DSSSL: *-то же самое-*);
- external-graphic-id (*id внешнего графического изображения*);
- graphic-max-height (*максимальная высота графического изображения*);
- graphic-max-width (*максимальная ширина графического изображения*);
- graphic-scale (*масштаб графического изображения*);
- indent-end (*правая граница абзаца*) (DSSSL: end-indent, CSS: -object-margin-);
- indent-start (*левая граница абзаца*) (DSSSL: start-indent, CSS: -object-margin-);
- inhibit-wrap (*запрет возврата*);
- keep (*привязать*) (DSSSL: *-то же самое-*);
- keep-with-previous (*привязать к предыдущему*) (DSSSL: *-то же самое-*);
- keep-with-next (*привязать к последующему*) (DSSSL: *-то же самое-*);
- position-point-x (*положение по оси x*);
- position-point-y (*положение по оси y*);
- position-reference (*опорное положение*) (DSSSL: *-то же самое-*);
- space-after-maximum (*пространство после максимума*) (DSSSL: space-after). *Не основное*;
- space-after-minimum (*пространство после минимума*) (DSSSL: space-after). *Не основное*;
- space-after-optimum (*пространство после оптимума*) (DSSSL: space-after);
- space-before-maximum (*пространство перед максимумом*) (DSSSL: space-before). *Не основное*;
- space-before-minimum (*пространство перед минимумом*) (DSSSL: space-before). *Не основное*;
- space-before-optimum (*пространство перед оптимумом*) (DSSSL: space-before);
- writing-mode (*режим записи*) (DSSSL: *-то же самое-*).

Обработка пробельных литер

Таблицы стилей XSL представляют собой XML-документы. По умолчанию пробельные литеры в языке XML сохраняются. Это может быть подчеркнуто в отдельных таблицах стилей с помощью элемента `xsl:preserve-space` в XSL или для отдельных элементов с помощью атрибута `xml:space` со значением `'preserve'` (сохранить).

В то же время элемент `xsl:stylesheet` может иметь атрибут `default-space`, который принимает или значение `'strip'` (удалить), или значение `'preserve'` (сохранить). Значение `'strip'` по умолчанию устанавливает удаление пробельных литер в результирующем документе. Если в элементе `xsl:stylesheet` атрибут отсутствует, то подразумевается значение по умолчанию `'preserve'`.

Если значение для атрибута `default-space` элемента `xsl:stylesheet` (установленное явно или по умолчанию) равно `'preserve'`, то для каждого отдельного результирующего элемента может быть использован элемент `xsl:strip-space`, что позволит удалить этот результирующий элемент из списка элементов, для которых сохраняются пробелы.

Пространство имен CSS

Другие пространства имен также могут предоставлять модели форматирования. Интересное предложение по использованию свойств таблиц стилей CSS для выстраивания потоковых объектов можно найти по адресу:

http://www.w3.org/TR/NOTE-XSL-and-CSS

Описание result-ns

Если есть только одно пространство имен, то есть пространство имен `fo:`, результирующее дерево описывается так:

```
<xsl:stylesheet
   xmlns:xsl="http//www.w3.org/TR/WD-xsl"
   xmlns:fo="http//www.w3.org/TR/WD-xsl/FO"
   result-ns="fo">
```

В данный момент не ясно, будет ли обеспечена возможность использовать два различных пространства имен в одной и той же таблице стилей, однако реальных оснований для подобного запрещения не существует. Два форматирующих пространства имен могут быть описаны следующим образом:

```
<xsl:stylesheet
xmlns:xsl="http//www.w3.org/TR/WD-xsl"
xmlns:fo="http//www.w3.org/TR/WD-xsl/FO"
xmlns:css=http://www.w3.org/TR/NOTE-XSL-and-CSS"
result-ns="fo"
result-ns="css">
```

Рассмотрим вкратце, как авторы спецификации CSS Хэйкен Ли (Hakon Lie) и Берт Бос (Bert Bos) предлагают использовать форматирующие объекты CSS*.

Основное правило шаблона с его *образцом* и *действием* применяется по-прежнему. Ниже приведено основное правило шаблона, которое использует пространство имен CSS:

```
<template match="partnumber">
   <css:chunk display="block"
              font-weight="bold"
              margin-top="20px">
      <css:chunk display="inline"
```

```
                    color="red">
                Part number:<css:space/>
            </css:chunk>
            <css:chunk color="green">
                <process-children/>
            </css:chunk>
        </css:chunk> <!-- Конец блока. -->
    </template>
```

Обратите внимание на использование элемента `css:chunk` для создания потокового объекта.

Свойства CSS преобразуются в формат XSL (с помощью простого метода преобразования, с которым мы встречались ранее в разделе «Преобразование CSS в XSL») и применяются как атрибуты элемента `css:chunk`.

Результатом указанного выше примера будет красный шрифт выводимой строки `"Part number:"`, за которой будет следовать содержание элемента `partnumber`, выделенное зеленым цветом.

Для дальнейшего обсуждения пространства имен CSS обращайтесь к сообщению (хорошо написанному!) консорциума W3C, упомянутому в начале этого раздела.

Данное пространство имен особенно интересно тем, что, скорее всего, «большой двойке» (компаниям Microsoft и Netscape) будет легче всего реализовать в своих браузерах именно этот вариант. Четвертые версии браузеров уже обеспечивают значительную поддержку таблиц стилей CSS. Можно с большой долей уверенности предположить, что инструмент для этого пространства имен появится ранее, чем инструмент для пространства имен `fo:`.

Другим интересным моментом может считаться, что уже существуют хорошо подготовленные программисты, для которых не составит труда использовать объекты CSS в XSL, а имеющиеся средства редактирования CSS могут быть легко преобразованы для поддержки этого формата.

Будущее языка XSL

Применяемая в настоящее время спецификация XSL оказалось не свободна от недостатков. Наиболее заметными из них, по общему мнению, являются отсутствие поддержки скриптов и табличного потокового объекта, а также отсутствие поддержки форм. Однако скорее всего эти пробелы будут восполнены в ближайших версиях рабочего проекта.

Окончательный вариант спецификации XSL будет представлять собой поистине монументальное произведение. В качестве *минимальной* цели авторы намерены объединить все функции таблиц стилей CSS и DSSSL! Таким образом, в конце концов, мы должны получить очень мощный и, возможно, очень сложный инструмент.

В связи с этим возникают следующие проблемы:

- найдется ли место и время для создания хорошего и дешевого программного обеспечение для поддержки подобной спецификации?
- будет ли спецификация достаточно проста, чтобы средний программист XML мог использовать ее вместо, скажем, CSS?

Если ответ на оба эти вопроса будет положительным, то будущее безусловно за такой спецификацией.

Если ответ хотя бы на один вопрос окажется отрицательным (а оба они тесно связаны друг с другом), то XSL станет всего лишь еще одним языком программирования, поддерживаемым малочисленными энтузиастами.

Одно из потенциальных применений дополнительных возможностей управления деревом документа упоминается на сайте консорциума W3C в разделе, посвященном информации, имеющей отношение к XSL (http://www.w3c.org/Style/xsl). Там предлагается использование таблиц стилей XSL в сочетании с CSS для обеспечения более тщательного контроля над результатом действия приложений. Это означает, что в некоторых более сложных приложениях может оказаться полезным «использование языка XSL на сервере для сжатия или настройки некоторых данных XML в более простой XML-документ, а затем применение таблиц стилей CSS на стороне клиента».

Заключение

В этой главе мы рассмотрели основы нового языка стилей XSL. Мы узнали:

- как язык XSL производит результирующее дерево из исходного дерева;
- что правило шаблона XSL имеет *образец* и *действие*;
- как сопоставлять шаблоны и узлы исходного дерева;
- как использовать пространства имен для идентификации форматирующих объектов;
- как задавать форматирующие объекты, применять к ним стили и затем их обрабатывать.

В конце мы познакомились с новой идеей использования пространства имен CSS для форматирования.

Глава 9. XML и уровни данных

Обратите внимание *Перед запуском примеров приложений настоятельно советуем проверить, не появились ли на нашем Web-сайте изменения к данной главе.*

С изменениями, а также с файлами, содержащими примеры приложений, можно познакомиться на нашем Web-сайте:

http://webdev.wrox.co.uk/books/1525/.

Язык XML является быстро развивающейся областью программирования, особенно активно этот процесс шел во время подготовки книги. В то время как теория, описанная здесь, остается все в тех же теоретических рамках, общедоступные компоненты анализаторов, которые мы использовали, могли быть изменены в соответствии с изменениями стандартов XML. Например, названия свойств и методов вряд ли останутся такими же, какие указаны в данной главе.

Во избежание разочарований, прежде чем устанавливать и запускать приложение, проверьте, пожалуйста, нет ли на нашем Web-сайте дополнений к нему.

В последнее время разработчики Web-сайтов много спорили по поводу судьбы HTML. В конце концов большинство специалистов сошлось на том, что XML скорее рано, чем поздно заменит его. В общем, к этому есть все основания. В то время как HTML представляет данные последовательным способом, XML их *описывает*. Таким образом, сила XML заключается в его способности передавать во внешние системы данные среднего уровня (Middle Tier), также называемого бизнес-уровнем (Business Tier), или уровни данных (Data Tiers) n-уровневой архитектуры без настройки программного обеспечения под каждого отдельного клиента. XML, созданный как текстовый язык для описания данных, позволяет обмениваться информацией между компьютерами вне зависимости от их операционных систем. Использование XML в качестве простой системы преобразования данных открывает возможность их полноценной обработки в Web между разнотипными платформами.

Передача сведений через Internet должна осуществляться не только между Web-браузерами и Web-серверами. Обмен деловой информацией может происходить между двумя серверами, автоматизирующими обработку заказов, оформление отчетов или перевод денег, делая реальными те функции, которые на основе Internet обещал обеспечить электронный обмен данными (Electronic Data Interchange, EDI). Благодаря широчайшей распространенности Internet, его вы-

сокой эффективности при обмене данными и нарождающемуся стандарту XML, специальные XML-приложения вскоре вполне могут заменить EDI.

В этой главе основное внимание уделено передаче данных клиентам любых типов (Web-браузерам, другим серверам и т.п.) при помощи универсального механизма XML и с использованием опыта и инструментов, к которым уже привыкли Web-разработчики. Здесь также описывается отсылка данных клиентам с использованием файлов, созданных Microsoft SQL Server, *хранимыми процедурами SQL* (SQL stored procedures) и *активными страницами сервера* (Active Server Pages). Особое внимание уделяется обработке данных в SQL Server, при этом показывается практически, как XML-обработка может охватить уровень клиента (Client Tier), средний уровень и уровни данных многоуровневой архитектуры. Это распределение нагрузки используется для максимального повышения эффективности системы и улучшения взаимодействия с пользователем. В этой главе будет показано, как при помощи обработки XML-данных на Web-сервере защитить компьютерные средства серверной стороны от прямого доступа к уязвимым ресурсам и скрыть топологию вашей Web-конфигурации.

Методы доставки XML-документов

В зависимости от приложения, существуют два способа использования XML-данных в качестве коммуникационной платформы между клиентом и сервером:

- использование механизма базы данных для создания XML-данных;
- использование промежуточных (middleware) систем для преобразования из XML в существующую инфраструктуру и обратно.

Каждый из способов имеет свои преимущества и недостатки. Обработка данных в промежуточных программах может обеспечить общий доступ ко многим системам, управляемым или не управляемым базами данных. Промежуточные программы также способны провести дополнительную обработку информации в процессе преобразования. Однако создание промежуточных программ может оказаться весьма дорогостоящей задачей, невыгодной для специальных приложений. Для редко меняющихся данных они также могут проигрывать в скорости обработки по сравнению с XML, созданным базами данных.

С другой стороны, XML, организованный механизмом базы данных, может включать в себя только те сведения, которые содержатся в этих базах данных. С его помощью значительно сложнее гибко представлять некоторые виды сведений, поскольку подобная методика оперирует индивидуальными рядами, а не иерархически выстроенной информацией. Однако механизм реляционной базы данных позволяет практически моментально выполнять обработку сложных источников информации, связанных через базу данных. Для редко меняющегося набора сведений это самый быстрый и удобный способ, поскольку XML-данные формируются только после изменений, а не каждый раз при запросе клиента. Если ваша задача – вскрытие информации в данных (data mining), единственным решением является формирование XML-данных на SQL-сервере, поскольку в промежуточных системах существует мало средств для OLAP-обработки, если таковые вообще существуют.

Уточнение *В этой главе основное внимание обращается на использование Microsoft SQL Server 6.5 как информационного архива данных. Поэтому упоминание SQL-сервера подразумевает именно Microsoft SQL Server, а не какие-либо другие SQL-серверные приложения, типа Sybase's SQL Server или базы данных Oracle.*

В этой главе будет также показано, как создавать XML-файлы, используя модули Web Task (Web Tasks) SQL-сервера для образования статических файлов, как организовывать хранимые процедуры SQL для динамического формирования моментального состояния данных, как проводить обработку данных на среднем уровне (Middle Tier) для настраиваемого возвращения XML-данных и для получения корректировок от клиента. Ни один из способов доставки не показывает клиенту непосредственно SQL-сервер (см. схему на рис. 9.1). Все потоки идут через Web-сервер. Не предоставляя клиенту прямого доступа к серверу, можно обеспечить безопасность и гибкость при обновлении систем, работающих на сервере (бэк-офиса) без необходимости обновления программного обеспечения клиента; клиент может даже и не знать, что были произведены какие-либо изменения. Это исключительно удачное решение для организаций, постепенно наращивающих возможности своей системы, так как все программное обеспечение обновляется на среднем уровне или на уровнях данных, совершенно незаметно для клиентов. Подобный подход также позволяет модифицировать системы, построенные под специфические требования. Например, если определенный XML-документ запрашивается только в ночное время, группа информационных технологий (IT) данной организации может решить не создавать динамический запрос к базе данных, а формировать отчет по ночам при помощи модуля Web Task SQL-сервера, экономя таким образом ресурсы процессора.

На схеме (рис. 9.1) показан обобщенный вид системы клиентов и серверов, обращающихся к Web-серверу корпорации.

В следующих разделах можно будет познакомиться с другими способами доставки с уровня данных информации клиенту.

Электронный список телефонов

В корпорации каждый сотрудник обычно получает список или брошюру с телефонными номерами своих коллег. Когда прежний работник покидает фирму или нанимается новый, список телефонов переделывают и раздают вновь; старый список чаще всего отправляется в мусорную корзину. В этой главе в качестве примера программы рассматривается электронная версия подобного телефонного справочника. Номера, размещенные в базе данных, все сотрудники могут просмотреть через свои Web-браузеры. Справочник легко обновляется, и это при нулевых затратах на копирование и распространение. В зависимости от политики компании, дополнительным преимуществом является доступность этого документа для сторонних организаций: деловых партнеров, других отделов или даже дя клиентов.

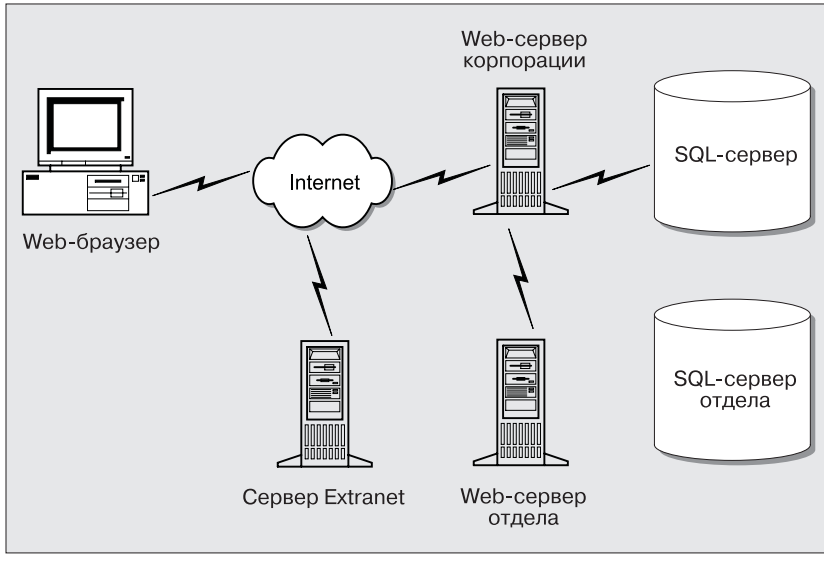


Рис. 9.1. Схема связи клиента с сервером

В приведенном ниже примере основное внимание уделяется отражению минимальной информации о сотрудниках. Подобная программа не предоставляет пользователю полный объем сведений, который руководство может посчитать необходимым. В ней содержится хранимый в базе данных список сотрудников, их телефонные номера и типы телефонов (факс, пейджер, сотовый телефон и т.д.). Вся информация помещается в четырех таблицах (рис. 9.2). Данные из этой программы-примера доставляются Web-браузеру при помощи XML и HTML. Для обновления справочника используются HTML-страница со списком сотрудников, HTML-форма создания запроса на обновление и страница Active Server Page, которая получает команды XML POSTs, изменяющие данные в таблицах. Некоторые XML-данные программа создает динамически, в ответ на каждый запрос из браузера, в то время как другие XML-данные она записывает в файл, чтобы показать, как используется модуль Web Task SQL-сервера. Это позволяет разгрузить систему, создавая XML-документы, доступные для клиента, во время минимальной загрузки системы, а не в часы пик. Следующий раздел посвящен способности SQL-сервера создавать файлы Web-типа.

Создание XML на SQL-сервере

Существует два различных способа создания XML в рамках SQL-сервера. Можно или использовать модули Web Task, которые являются автоматическими модулями, запускаемыми через определенные интервалы, и формируют либо XML-, либо HTML-файлы; или применить хранимые процедуры для возврата набора результатов, преобразуемых в XML. Первый подраздел посвящен обсуждению модулей Web Task, за ним следует раздел, описывающий использование хранимых процедур для возврата набораXML-результатов.

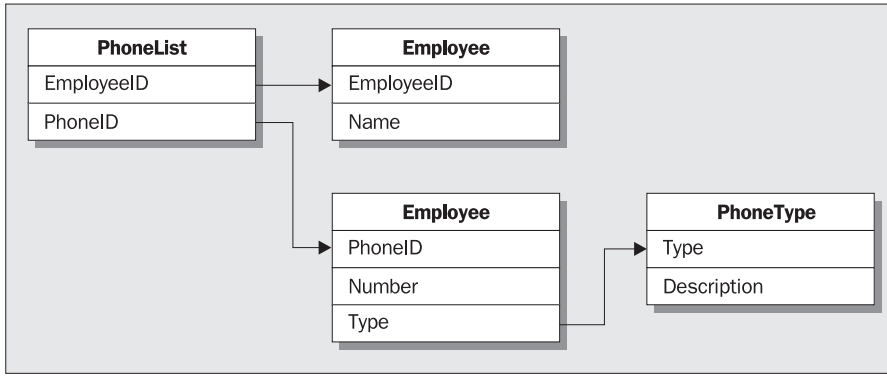


Рис. 9.2. Схема отношений между объектами телефонного списка

Модули Web Task для автоматического создания XML файлов

Microsoft's SQL Server имеет возможности, упрощающие организацию HTML-файлов из SQL-предложений. Обычно такая необходимость появляется при подготовке отчетов, которые размещаются на Web-сайте. Применение подобного способа имеет в основе внутреннюю IT-систему формирования такого рода документов, которая предоставляет равноправный доступ всем пользователям, поскольку единственным условием их получения является наличие Web-браузера. Прежние системы отчетов, напечатанные на бумаге в силу стоимости и т.п. имели ограниченное распространение. Сетевые технологии сделали организационную структуру более «плоской», доступ к информации, ранее предназначенной только для «привилегированных», сейчас зачастую осуществляется простым нажатием клавиши.

Так как отчеты подобных типов в подавляющем большинстве оформлены по стандарту, одинаковому для любого, кто желает их просмотреть, нет необходимости каждый раз создавать частное Web-приложение, которое интерактивно запрашивает данные и строит HTML-результаты. Все что действительно необходимо – это создать одну или несколько HTML-страниц в рамках одного сеанса работы. Именно по этой причине были разработаны модули Web Task. Каждая задача может быть выполнена через определенный временной интервал: ежедневные отчеты могут создаваться каждый вечер, еженедельные – по понедельникам, например, в пять утра; где-то могут потребоваться даже почасовые отчеты. Подобные задачи автоматически выполняются SQL-исполнителем (SQL Executive) и позволяют более эффективно использовать ресурсы, поскольку такие ситуации не требуют обмена информацией по сети. Можно так составить расписание, что выполнение задачи будет осуществляться до какой-то даты или пока не произойдет указанное число событий. В документ можно включить ссылки на другие отчеты. Также допустимо автоматическое создание HTML-кода в форме таблицы. Однако при этом не следует ожидать высокохудожественного дизайна. Чтобы получить действительно красиво оформленную HTML-страничку, необходимо предоставить SQL-серверу файл шаблона, куда он вставит данные, основываясь на ключевых словах, содержащихся в исходном HTML-коде.

До сих пор мы упоминали только HTML. Это происходит потому, что модули Web Task были разработаны как раз для формирования HTML. Когда такие модули были добавлены в SQL-сервер, спецификация XML еще находилась в стадии разработки. Подлинная гибкость работы с шаблонами на SQL-сервере проверяется на возможности его использования для генерации XML. Серверу безразлично, содержит ли файл что-либо помимо ключевых слов, которые он использует для вставки данных в текущий ряд. В приведенном ниже шаблоне нет ни одной строчки на HTML. Он скомпонован из тэгов XML-элементов, используемых для придания красивой формы документу, и ключевых слов `<%begindetail%>`, `<%enddetail%>` и `<%insert_data_here%>`, что означает: (начать элемент, закончить элемент и вставить_данные_здесь).

Ключевые слова `<%begindetail%>`, `<%enddetail%>` используются, чтобы отметить область, где следует использовать набор результатов данного SQL-запроса для выполнения инструкции `<%insert_data_here%>`.

Ключевое слово `<%insert_data_here%>` указывает, куда следует поместить данные в виде столбцов из набора результатов.

Для каждого столбца в наборе результатов допустимо только одно ключевое слово `<%insert_data_here%>`. Ниже приведен шаблон модуля Web Task, использованный для формирования XML-списка телефонов:

```
<EmployeeList>
<%begindetail%>
   <Entry>
       <Employee><%insert_data_here%></Employee>
       <Phone><%insert_data_here%></Phone>
       <Type><%insert_data_here%></Type>
   </Entry>
<%enddetail%>
</EmployeeList>
```

Многочисленные SQL-запросы могут использоваться для заполнения многочисленных пар `<%begindetail%>` и `<%enddetail%>`. Это позволяет создавать достаточно сложные формы отчетов. Одним из недостатков подхода на основе шаблона модуля Web Task является невозможность вложения пар `<%begindetail%>` и `<%enddetail%>` одна в другую, то есть:

```
<!-- Пример того, чего нельзя сделать при помощи шаблонов Web Task. -->
<EmployeeList>
<%begindetail%>
   <Entry>
       <Employee><%insert_data_here%>
       <%begindetail%>
       <Phone><%insert_data_here%></Phone>
           <Type><%insert_data_here%></Type>
       <%enddetail%>
       </Employee>
   </Entry>
<%enddetail%>
</EmployeeList>
```

Этот пробел способен в значительной степени разочаровывать пользователей в возможностях иерархической структуры XML. Если бы подобная операция являлась допустимой, можно было бы с легкостью формировать сложные XML-документы, которые использовали бы преимущества иерархического построения для сокращения лишней передачи данных и создания сложных объектных моделей. При данном ограничении на вложенность пар выходные данные нашего XML-шаблона содержат несколько элементов `<Employee></Employee>` (Сотрудник) для одного и того же сотрудника, в то время как одного элемента для них было бы вполне достаточно. Ниже даны результаты выполнения модуля Web Task, полученные с шаблоном `Phone List XML Template`:

```
<EmployeeList>

   <Entry>
       <Employee>John Doe</Employee>
       <Phone>555-1213</Phone>
       <Type>Working</Type>
   </Entry>

   <Entry>
       <Employee>John Doe</Employee>
       <Phone>555-1217</Phone>
       <Type>Home FAX</Type>
   </Entry>

   <Entry>
       <Employee>Janet Doe</Employee>
       <Phone>555-1213</Phone>
       <Type>Mobile</Type>
   </Entry>

   <Entry>
       <Employee>Jim Johnson</Employee>
       <Phone>555-1214</Phone>
       <Type>Cellular</Type>
   </Entry>

   <Entry>
       <Employee>Jim Johnson</Employee>
       <Phone>555-1213</Phone>
       <Type>Working</Type>
   </Entry>

   <Entry>
       <Employee>Sally Mae</Employee>
       <Phone>555-1215</Phone>
       <Type>Business FAX</Type>
   </Entry>

</EmployeeList>
```

Этот файл, а также другие файлы для данной главы могут быть загружены с нашего Web-сайта http://webdev.wrox.co.uk/books/1525. Мы уже говорили, как создать XML при помощи шаблона модуля Web Task, и о некоторых возможнос-

тях этого модуля; сейчас самое время перейти к рассмотрению вопроса, каким же образом формируется модуль Web Task.

Использование SQL Server Web Assistant

Автоматизированный модуль Web Task проще всего создать, воспользовавшись Web-помощником SQL-сервера (SQL Server Web Assistant). Эта возможность была осуществлена в Microsoft's SQL Server 6.5. В него входит *мастер* (wizard), который поэтапно проводит пользователя через процесс создания модуля Web Task, скрывая за графическим интерфейсом пользователя (Graphical User Interface (GUI)) все исполняемые рутинные операции. Первым этапом в создании модуля Web Task является указание, какой SQL-сервер должен быть запущен. Поскольку этот этап работы мастера очевиден, мы не будем его здесь описывать. Следующим, действительно интересным и важным этапом является указание SQL-запроса, который возвратит XML-данные.

Это может быть сделано либо графически, либо путем набора текста запроса, либо с использованием хранимой процедуры; последний способ иллюстрируется рис. 9.3.

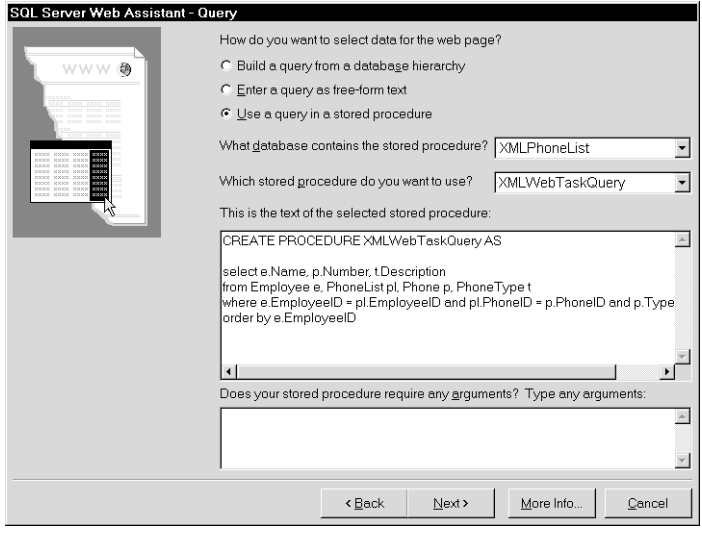


Рис. 9.3. Указание SQL-запроса для возврата XML-данных

Мастер позволяет ввести любые специальные параметры, которые могут понадобиться хранимой процедуре. Заметьте, что базу данных и запрос допускается выбрать из раскрывающегося списка. Это очень полезное свойство, особенно если вы не помните точного имени или правильного написания имени хранимой процедуры, котоую собираетесь использовать.

На следующем этапе мастер попросит указать расписание запуска создаваемого модуля. В раскрывающемся списке демонстрируются имеющиеся в рамках SQL-докуента варианты (см. рис. 9.4).

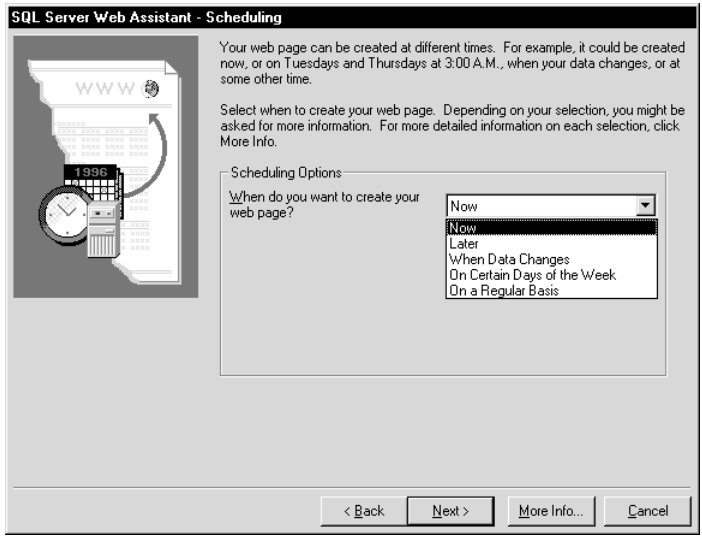


Рис. 9.4. Ввод расписания запуска создаваемого модуля Web Task

При выборе любой опции, отличной от *сейчас* (Now), появляется дочернее диалоговое окно, который позволяет указать, когда следует запускать модуль: дни, часы и т.п.

Следующий этап, описываемый в этой главе, связан с выбором шаблона и файла вывода результатов. Шаблон, выбранный на рис. 9.5, будет использоваться на всем протяжении данной главы. Файл вывода может иметь любое имя, в нашем примере, его мя созвучно с именем шаблона.

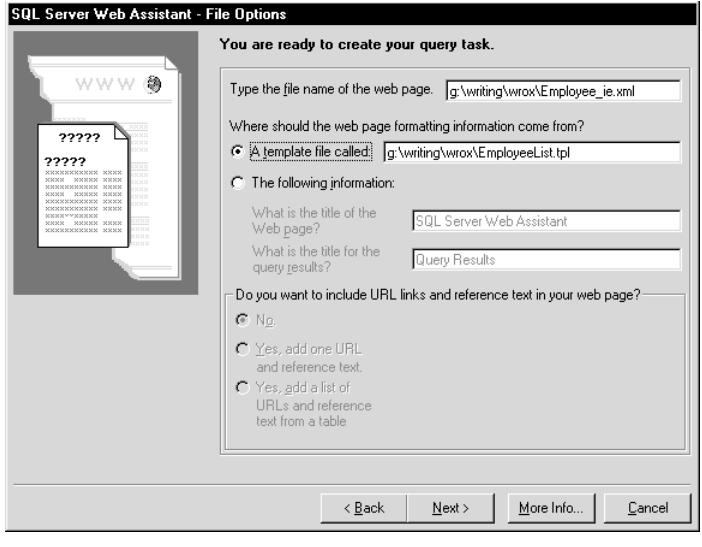


Рис. 9.5. Выбор шаблона и имени файла вывода для создаваемого модуля Web Task

Следующие этапы построения при помощи мастера связаны с настройкой HTML-результатов. Поскольку сейчас нашей целью является получение XML-данных, а не создание красивой Web-странички, эти этапы мы опустим. Поскольку построение выбирается на основе шаблона, все установки красочного представления данных, которые можно было бы ввести на данных этапах, все равно будут проигнорированы.

Web-помощник SQL-сервера – простой способ быстрого создания модуля Web Task. Однако поскольку все операции «спрятаны» под GUI, он не дает возможности разобраться, что, как и почему работает. Следующий раздел детально раскрывает эти вопросы.

Создание модулей Web Task с использованием SQL

SQL-сервер предоставляет системную хранимую процедуру `sp_makewebtask` для создания модуля Web Task. Этот механизм вызывает расширенную хранимую процедуру, которая находится в DLL, хотя в ней может быть больше информации, чем вам требуется. Если указать параметры периодичности, задача будет добавлена в таблицу системных операций и может быть просмотрена при помощи пункта меню **Server ⇒ Scheduled Tasks**... (Сервер ⇒ Задания по расписанию...) менеджера предметной области (SQL Enterprise Manager). Использование процедуры `sp_makewebtask` для создания текущего мдуля Web Task показано на рис. 9.6.

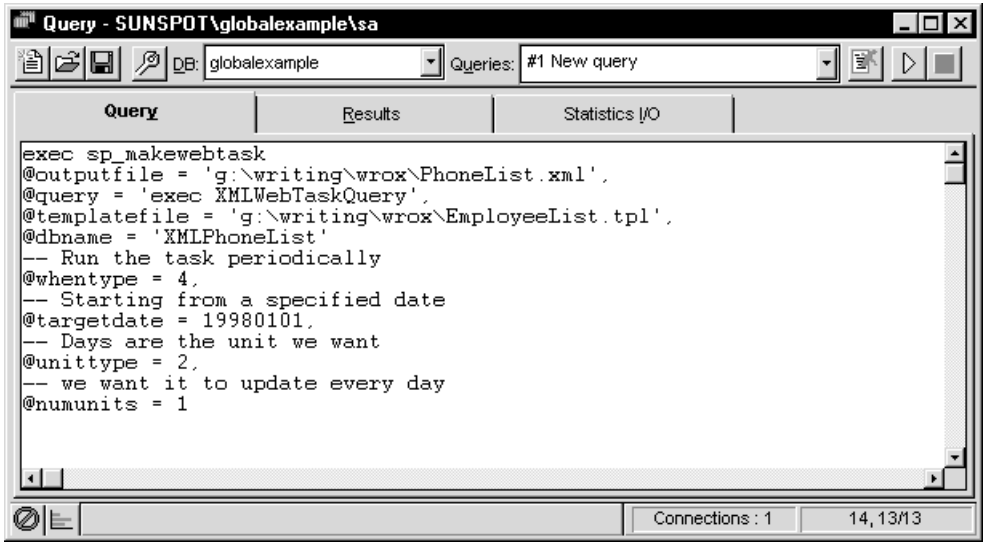


Рис. 9.6. Использование процедуры sp_makewebtask

При использовании этого способа следует указать запрос, который необходимо выполнить, имя базы данных, где находится таблица, имя выходного файла и имя файла шаблона. Оставшиеся параметры хранимой процедуры относятся к вводу временных параметров запуска модуля, внесенного в расписание.

Параметр `@targetdate` указывает дату запуска модуля. Учтите, что дата представляется не в стандартном формате данных `datetime`, а целым числом. Для даты предусмотрен следующий формат: `yyyymmdd` (`yyyy` – год, `mm` – месяц, `dd` – день).

Оставшиеся параметры используются для указания периодичности запуска модуля:

- `@whentype` – параметр установлен на создание XML файла каждые *n* минут, часов, дней или недель, в зависимости от параметра `@unittype` (который в нашем примере равен 2). Дополнительная информация о типах `@whentype` находится в Transact SQL Help;
- `@unittype` – возможные опции: 1 – для часов, 2 – для дней, 3 – для недель, 4 – для минут;
- `@numunits` – окончательный параметр, описывающий периодичность выполнения задачи; указывает количество единиц времени, которые должны быть пропущены до следующего запуска модуля.

Например, если `@unittype` равен 4, а `@numunits` равен 15, то модуль Web Task будет запускаться каждые 15 минут. Использование процедуры `sp_makewebtask` обеспечивает большую гибкость в работе. Вместе с гибкостью, однако, обнаруживается и сложность подобного метода. Более простой способ организации периодически запускаемого модуля Web Task – это использование диалогового окна **New Task** в менеджере предметной области (**SQL Enterprise Manager's New Task**) (рис. 9.7), который можно вызвать черз пункты меню: **Server** ⇒ **Scheduled Tasks**.

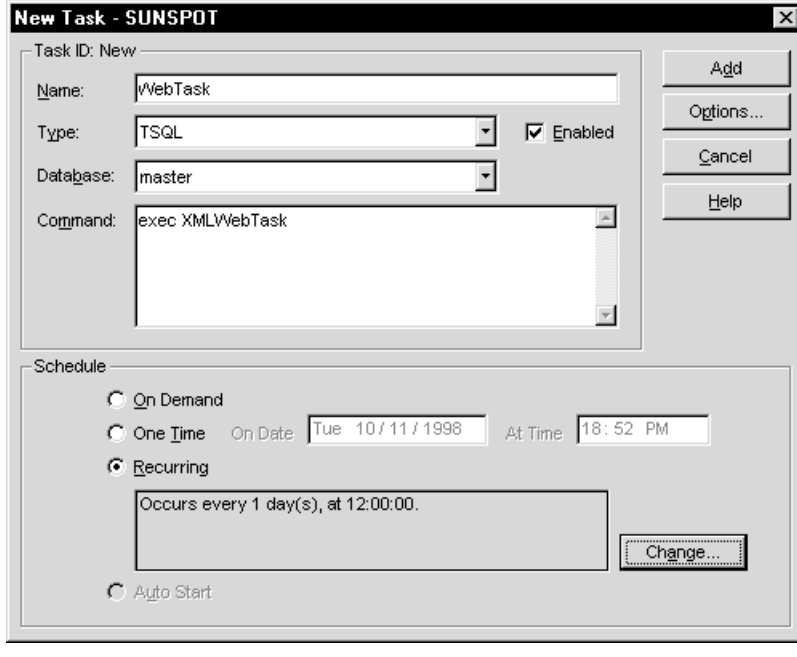


*Рис. 9.7. Диалоговое окно **New Task***

В нашем примере процедура `sp_makewebtask` не вызывается непосредственно, вместо нее появляется хранимая процедура, которая была создана для использования вместе с триггерами. Она, в свою очередь, вызывает процедуру `sp_makewebtask`, инициируя ее немедленный однократный запуск. Указать периодичность запуска

при помощи GUI проще, чем пытаться запомнить все параметры и их значения в зависимости от содержания. Общий итог одинаков: XML-файл будет создаваться автоматически, с указанной периодичностью.

Триггеры и модули Web Task

Порой случается, что ежедневные или даже ежечасные отчеты не удовлетворяют всех потребностей пользователей. Подобные официальные сообщения должны предоставлять пользователям самую свежую информацию, невзирая на то, что все они просматривают документ без настроек программы, изменения критериев поиска или временных рамок. Даже с учетом этих требований неразумно затрачивать усилия и время процессора на построение динамических отчетов, используя страницы Active Server Pages и хранимые процедуры. Но как же собрать самую свежую информацию, если данные меняются спорадически и непоследовательно? Ответ – с помощью триггеров.

Триггеры – это по сути дела хранимые процедуры, которые запускаются для указанной таблицы при наступлении определенных событий. События – это `Insert`, `Uptade` или `Dedat` (Вставить, Обновить, Удалить). Можно выполнять SQL-код для одного, двух или для всех трех событий. Можно выполнять разные коды для разных событий. Использование триггеров – это самый верный способ запуска обработки данных при наступлении события «изменение данных». На рис. 9.8 представлен триггр `PhoneType` (Тип номера) модуля Web Task.

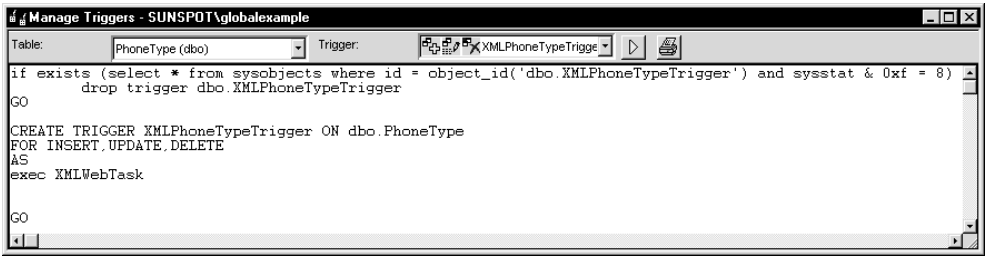


Рис. 9.8. Триггер PhoneType модуля Web Task

Чтобы XML-данные отражали самую свежую информацию, обычно пользуются набором триггеров, которые постоянно перезаписывают файл. Поскольку обработка для событий `Insert`, `Update` и `Delete` одинакова, каждый триггер обрабатывает все три события. На рис. 9.8 показан триггер, использованный для отслеживания изменений в таблице `PhoneType`. Этот триггер вызывает хранимую процедуру `XMLWebTask`, чтобы заново сформировать XML-данные. Он использует тот же однократно запускаемый модуль Web Task, уже описанный в этой главе. Каждый триггер имеет уникальное имя и основан на определенной таблице. Как видно из приведенного ниже SQL-кода, все триггеры производят одну и ту же обработку данных, имеют однотипную маску событий (`Insert`, `Updat`, `Delete`) и им присвоены имена, которые указывают на таблицу, с которой триггеры ассоциированы, и на тот факт, что триггеры создают XML-файлы. Это сделано для облегчения понимания того, какие именно операции выполняются этими триггерами, при их просмотре в окне, вызываемом через меню **Manage** ⇒ **Triggers**... (Управление ⇒ Триггеры) в SQL Enterprise Manager's...GUI. Ниже дан SQL-код

для создания триггеров. Он может быть загружен с нашего Web-сайта http://webdev.wrox.co.uk/books/1525:

```
if exists (select * from sysobjects where id = object_id('dbo.XMLEmployeeTrigger')
and sysstat & 0xf = 8)
   drop trigger dbo.XMLEmployeeTrigger
GO

CREATE TRIGGER XMLEmployeeTrigger ON dbo.Employee
FOR INSERT,UPDATE,DELETE
AS
exec XMLWebTask
GO

if exists (select * from sysobjects where id = object_id('dbo.XMLPhoneTrigger')
and sysstat & 0xf = 8)
   drop trigger dbo.XMLPhoneTrigger
GO

CREATE TRIGGER XMLPhoneTrigger ON dbo.Phone
FOR INSERT,UPDATE,DELETE
AS

exec XMLWebTask

GO

if exists (select * from sysobjects where id = object_id('dbo.
XMLPhoneListTrigger') and sysstat & 0xf = 8)
   drop trigger dbo.XMLPhoneListTrigger
GO

CREATE TRIGGER XMLPhoneListTrigger ON dbo.PhoneList
FOR INSERT,UPDATE,DELETE
AS

exec XMLWebTask

GO

if exists (select * from sysobjects where id = object_id('dbo.
XMLPhoneTypeTrigger') and sysstat & 0xf = 8)
   drop trigger dbo.XMLPhoneTypeTrigger
GO

CREATE TRIGGER XMLPhoneTypeTrigger ON dbo.PhoneType
FOR INSERT,UPDATE,DELETE
AS

exec XMLWebTask

GO
```

Чтобы до конца разобраться в том, что происходит при срабатывании триггера, следует изучить код хранимой процедуры. Приведенный ниже код применяется для создания модуля Web Task (хранимая процедура `XMLWebTask`), который запускается немедленно, исполняется один раз (непериодично) и использует уже описанный нами ранее шаблон.

```
if exists (select * from sysobjects where id = object_id('dbo.XMLWebTask') and
sysstat & 0xf = 4)
   drop procedure dbo.XMLWebTask
GO

CREATE PROCEDURE XMLWebTask AS
exec sp_makewebtask @outputfile = 'g:\writing\wrox\PhoneList.xml',
@query = 'exec XMLWebTaskQuery',
@templatefile = 'g:\writing\wrox\EmployeeList.tpl',
@dbname = 'XMLPhoneList'
GO
```

Хранимая процедура `XMLWebTask` в свою очередь указывает на другую хранимую процедуру `XMLWebTaskQuery`, которая возвращает подмножество столбцов таблиц, связанных с использованием соответствующих ключей. Код процедуры `XMLWebTaskQuery` выглядит следующим образом:

```
if exists (select * from sysobjects where id = object_id('dbo.XMLWebTaskQuery')
and sysstat & 0xf = 4)
   drop procedure dbo.XMLWebTaskQuery
GO

CREATE PROCEDURE XMLWebTaskQuery AS

select e.Name, p.Number, t.Description
from Employee e, PhoneList pl, Phone p, PhoneType t
where e.EmployeeID = pl.EmployeeID and pl.PhoneID = p.PhoneID and p.Type = t.Type
order by e.EmployeeID
GO
```

Вы, возможно, уже догадались, что создание XML-файлов при помощи триггеров – самый надежный способ растратить ресурсы процессора, если данные в таблицах часто меняются. Когда поток изменений в таблицах велик, ресурсы системы используются для создания сотен (а то и тысяч) XML-файлов. Если изменения происходят группами, это не проблема. Если же приложение постоянно обновляет базу данных, лучше воспользоваться страницами Active Server Pages и хранимыми процедурами для построения XML-данных «на лету». Хотя этот подход интенсивнее использует процессор, нагрузка распределяется между двумя и более машинами, и общая нагрузка на SQL-сервер окажется меньше. Безусловно, приложения могут налагать такие высокие требования, что реализация этого подхода окажется слишком сложной. В подобных случаях можно занести в расписание модуль Web Task, который запускался бы ежеминутно. Выбор между триггерами и периодически запускаемыми модулями Web Task приходится делать для каждого конкретного приложения.

До сих пор все описанное в этой главе касалось создания XML-данных. Теперь, когда данные уже размещены где-либо на сервере, необходимо, чтобы клиент загрузил их и вывел на экран пользователя. В следующем разделе детально обсуждаются HTML и JavaScript, необходимые для представления XML-данных в виде таблицы на экране пользователя.

HTML-страница для вывода XML-файла

Сейчас, когда мы различными путями научились создавать XML-файлы, пора познакомиться с тем, как найти XML-файл и показать его пользователю. В предыдущих главах было рассказано о Microsoft's XML-анализаторе, то есть таком механизме, который доставляет URL в XML источник и разбирает XML-элементыв объектную модель. Код JavaScript использует анализатор MSXML для построения таблицы и ее прикрепления к элементу `SPAN` на HTML-странице. Все это происходит во время события `onLoad` (при загрузке) тэга `BODY` (тело), так что в результате таблица, созданная при помощи XML, появляется одновременно с другими HTML-элементами.

На рис. 9.9 представлен файл `Phoneist.xml`, сформированный с применением модуля Web Task.

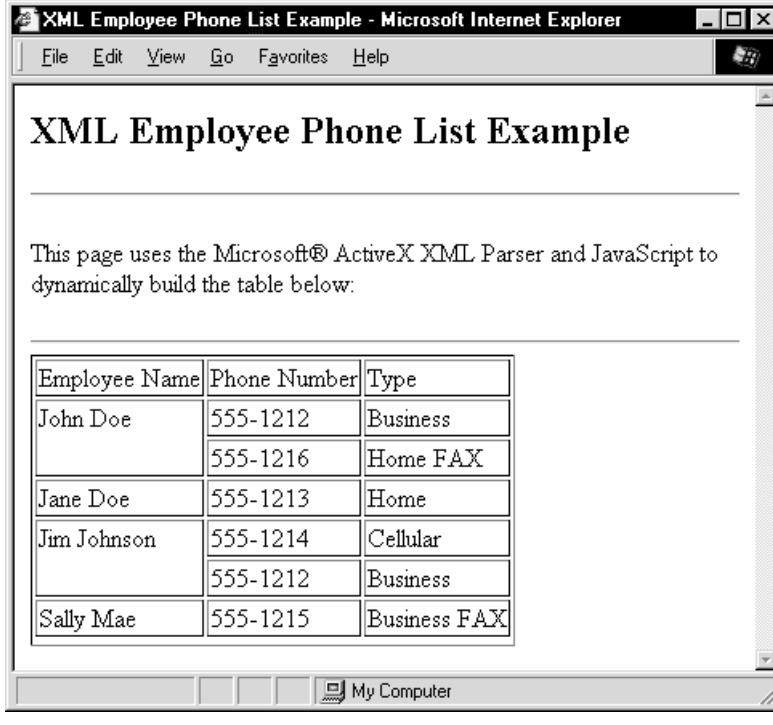


Рис. 9.9. Отображение файла PhoneList.xml

Это явно не самая красивая HTML-страница, но цель добиться эстетического совершенства и не ставилась. Действительной ценностью является код, использованный для создания таблицы. Первое, что он делает – сообщает анализатору MSXML, откуда следует получить данные; эта операция осуществляется в функции `OnFetchXmlData()`, приведенной ниже. URL для данных должен происходить из того же домена и директории, что и HTML-страница, в противном случае пользователь сразу обнаружит ошибку сценария. Функция `ResolveURL()` как раз несет ответственность за то, чтобы запрос на XML-данные пришел из того же домена и директории, где находится HTML-страница. Ниже приведен код для `EmployeePhoneList.htm`.

Полностью файл может быть загружен с нашего Web-сайта: http://webdev.wrox.co.uk/books/1525.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">

<HTML>
<HEAD>
   <TITLE>XML Employee Phone List Example</TITLE>
</HEAD>

<SCRIPT LANGUAGE="JavaScript">
   //<!--

   // Контейнер для Xml-документа.
   var XmlDoc;

   function ResolveURL(url)
   {
       var loc = window.location.toString();
       var i = loc.lastIndexOf("/");
       var result = loc.substring(0,i+1) + url;
       return result
   }
   function OnFetchXmlData()
   {
       // Создаем экземпляр XML-анализатора.
       XmlDoc = new ActiveXObject("msxml");
       // Приказываем анализатору получить данные с сервера.
       XmlDoc.url = ResolveURL("PhoneList.xml");
       // Ожидаем, пока окончится передача и разбор.
       window.setTimeout("CheckProgress()",100);
   }
```

В код введена пауза, за время которой считываются и обрабатываются данные. При этом происходит постоянный опрос состояния анализатора, пока файл не будет полностью обработан; далее вызывается функция построения таблицы `BuildXmlTable()`.

```
   function CheckProgress()
   {
       var state = XmlDoc.readyState;
       // Проверяем, закончена ли работа анализатора.
       if( state == 0 || state == 1 || state == 2 || state == 3)
       {
           // Отключаемся ненадолго и проверяем снова.
           window.setTimeout("CheckProgress()",100);
       }
       else if( state == 4 )
       {
           // Все разобрано, теперь строим код HTML..
           // ..чтобы представить данные в виде таблицы.
           BuildXmlTable();
       }
       else
```

```
        {
            alert("Failed to load XML data");
        }
    }
```

Следующая функция создает описание HTML-таблицы «на лету», вычисляя ROWSPAN для каждого сотрудника. Эти вычисления не потребовались, если бы мы могли формировать XML-данные несколько иначе, что, как уже упоминалось, невозможно в силу ограничений на вложенность в модулях Web Task. Вместо этого данные анализируются на уровне клиента и эффективно преобразуются в показанную ранее структуру, состоящую из одного элемента «сотрудник» и нескольких элементов «телефон».

```
    function BuildXmlTable()
    {
        // Запускаем таблицу.
        var tableHTML = "<TABLE BORDER BORDERCOLOR=#000000><tr><td>Employee
        Name</td><td>Phone Number</td><td>Type</td></tr>";
        // Устанавливаем корень дерева XML <ITEMLIST>.
        var root = XmlDoc.root;
        var nIndex = 0;
        // Получаем массив дочерних пунктов.
        var arrItems = root.children;
        // Текущая запись телефонного списка.
        var entry = null;
        var nRowSpan = 0;
        // Проходим циклически по всем <Entry> в <EmployeeList> и добавляем их..
        // ..к таблице.
        for( nIndex = 0; nIndex < arrItems.length; nIndex++ )
        {
            // Получаем объект <Entry> по данному указателю.
            entry = arrItems.item(nIndex);
            if( nRowSpan == 0 )
            {
        // Вычисляем количество строк.
        nRowSpan = CalcRowSpan(nIndex,arrItems);
        // Теперь строим код HTML для данного входа.
        tableHTML = tableHTML + "<tr><td VALIGN=TOP rowspan=" + nRowSpan + ">";
        tableHTML = tableHTML + entry.children.item("Employee").text + "</td>";
        }
        tableHTML = tableHTML + "<td>" + entry.children.item("Phone").text + "</
td>";
        tableHTML = tableHTML + "<td>" + entry.children.item("Type").text + "</
td>";
            // Закрываем эту строку таблицы.
            tableHTML = tableHTML + "</tr>";
            // Уменьшаем счетчик строк так, чтобы он в конце концов обнулился.
            nRowSpan--;
        }
        // Закрываем тэг таблицы.
```

```
        tableHTML = tableHTML + "</TABLE>"
        // Показываем созданную таблицу.
        EmployeeListSpan.innerHTML = TableHTML;
    }

    function CalcRowSpan(nStartIndex,arrEntries)
    {
        // Сохраняем имя сотрудника.
        var strName = arrEntries.item(nStartIndex).children.item("Employee").text;
        // Разгружаем переменные контейнера.
        var strCurrName = strName;
        var nCount = 0;
        var nIndex = nStartIndex + 1;
        // Проходим циклом по массиву, подсчитывая вхождения имени сотрудника.
        while( nIndex < arrEntries.length && strName == strCurrName )
        {
            // Получаем имя следующего сотрудника
            strCurrName = arrEntries.item(nIndex).children.item("Employee").text;
            // Увеличиваем указатель и счетчик rowspan.
            nIndex++;
            nCount++;
        }
        return nCount;
    }

    //-->
</SCRIPT>

<BODY BGPROPERTIES="FIXED" BGCOLOR="#FFFFDD" ONLOAD="OnFetchXmlData()">

<H2>XML Employee Phone List Example</H2>

<HR>

<P>
This page uses the Microsoft&reg; ActiveX XML Parser and JavaScript to<br>
dymamically build the table below:
</P>

<HR>

<SPAN ID="EmployeeListSpan"></SPAN>

</BODY>
</HTML>
```

Создание XML с помощью хранимых процедур

Как уже упоминалось в этой главе, существует два способа, с помощью которых SQL-сервер может формировать XML-данные. Первый, уже изученный нами, – это использование модуля Web Task. Второй – использование хранимой процедуры, которая выдает набор результатов в виде XML-данных. Это хороший способ формирования самых свежих наборов XML-данных на Web-сервере, позволяющий обходиться без серьезных издержек. Единственная специальная обработка происходит на SQL-сервере, где хранимые процедуры уже были ском-

пилированы для большей эффективности. В среднем это сберегает до 30% мощности процессора, так как SQL-обработка уже была выполнена. Все, что необходимо – обыкновенная страница ASP, которая передает XML-данные непосредственно запросившему их клиенту.

Подобное использование хранимых процедур потребует несколько большей изощренности, чем создание модуля Web Task. Модуль Web Task обрабатывает данные по одной записи, используя `<%insert_data_here%>`, в то время как набор результатов возвращается в виде их полного перечня, и обработка каждой отдельной записи при этом оказывается проблематичной. К счастью, курсоры, которые первоначально были включены в ODBC-библиотеки для SQL-сервера, добавлены и в SQL-сервер. Это позволяет манипулировать данными на уровне записей и включает возможность по мере поступления изменять или удалять их.

Тем не менее, даже с помощью названной функции создание XML-данных все равно остается непростой задачей. Одна из проблем заключается в невозможности создания временных переменных типа `text`. Переменные `chars` и `varchars` имеют предел длины (255 символов), который быстро исчерпывается при создании XML-данных. Предельная длина в 255 символов для `chars` и `varchars` изменяется до 8192 символов в SQL-сервере 7.0, но для большинства XML-приложений этого все равно очень мало. Таким образом, единственным выходом является создание временной таблицы с полями типа `text`. Временные таблицы часто формируются с использованием инструкции `SELECT...INTO` и запроса, вставляющего все записи, которые требуется поместить в таблицу; иные программисты предпочитают использовать инструкцию `CREATE TABLE` и затем инструкцию `INSERT`, утверждая, что при этом код хранимой процедуры становится более легким для понимания.

Любой из способов хорош.

Использование курсора для селективной вставки или управления данными, вносимыми во временную таблицу, придает работе особый интерес. В нашем примере требуется построить всего одну запись XML-данных, поскольку страница ASP используется только для сквозного прохода и не должна производить специальной обработки, например, соединения нескольких записей в одну строку.

Чтобы создать временную таблицу, хранимая процедура `GetXML` вставляет текстовую `(text)` версию открывающего корневого тэга, `<EmployeeList>`, в таблицу (см. код). Символ `#` указывает SQL-серверу, что эта таблица – локальная временная таблица, которая может быть удалена по завершении выполнения хранимой процедуры. Можно также создать глобальную временную таблицу, которая будет удалена по окончании сеанса работы, а не по завершении исполнения хранимой процедуры. Глобальные временные таблицы указываются знаком `##`. Когда запись вставлена, локальная переменная установлена на указатель `(pointer)` поля типа `text`, который является переменной `varbinary(16)`. Этот указатель используется для всех последующих манипуляций с записью типа `text`.

```
if exists (select * from sysobjects where id = object_id('dbo.GetXML') and sysstat
0xf = 4)
   drop procedure dbo.GetXML
GO

CREATE PROCEDURE GetXML AS
```

```
BEGIN

   -- Создаем временную таблицу для размещения данных XML.
   SELECT "XML" = CONVERT(text,'<EmployeeList>')
   INTO #Work

   DECLARE @@ptrval varbinary(16)
   -- Определяем указатель на столбец XML во временной таблице.
   SELECT @@ptrval = TEXTPTR(XML)
   FROM #Work
```

Следующим шагом является создание локальных переменных, для размещения записей, которые возвращает инструкция курсора `SELECT`. Курсор `WorkCursor` предназначен для итеративного прохода по записям набора результатов. К сожалению, невозможно указать хранимую процедуру для части набора результатов курсора, поэтому взамен используется внутренняя часть процедуры `XMLWebTaskQuery`.

```
DECLARE @Name varchar(255)
DECLARE @Number varchar(32)
DECLARE @Desc varchar(255)

-- Используем курсор для итеративного прохода по строкам с построением данных XML.
DECLARE WorkCursor CURSOR
FOR SELECT e.Name, p.Number, t.Description
   FROM EmployeeID = pl.EmployeeID and pl.PhoneID = p.PhoneID and p.Type = t.Type
       ORDER BY e.EmployeeID
FOR READ ONLY
```

Как и в группе записей, теперь до начала работы с данными следует перейти к первой записи курсора. Переменная `@@fetch_status` устанавливается на отображение того, успешно ли произошел этот переход. Когда используется оператор `FETCH`, SQL-сервер копирует `COLUMNAR` (колоночные) данные в три локальные переменные в том порядке, в каком столбцы располагались в наборе записей. Затем код использует данные столбцов, переменную текстового указателя и команду `UPDATETEXT`, чтобы добавить данные к одной из записей типа `text` в таблице `#Work`. При вызове команды `UPDATETEXT` переменная `@@ptrval` отождествляет указатель SQL-сервера с текстовым полем в таблице `#Work`. Две опции, следующие за указателем текста, показывают, что текст должен быть подсоединен к концу данных столбца (`NULL`) и что следует заменить ноль (`0`) символов. Заметим, что XML элементы циклически переносятся в данные в столбцах, по мере того как они вставляются во временную таблицу.

Этот процесс продолжается до тех пор, пока все записи в курсоре не будут обработаны, после чего курсор высвобождается. Последний шаг – возвращение данных вызывавшей их программе. Ниже приведен полный код хранимой процедуры `GetXML`, которая возвращает XML-данные; этот код может быть загружен с нашего Web-сайта http://webdev.wrox.co.uk/books/1525.

```
   -- Подготавливаем курсор к работе.
   OPEN WorkCursor
   -- Принимаем первую строку.
```

```
    FETCH NEXT FROM WorkCursor INTO @Name, @Number, @Desc
    WHILE @@fetch_status = 0
    BEGIN
        -- Добавляем к текстовому полю тэги XML и переменные столбцов.
        UPDATETEXT #Work.XML @@ptrval NULL 0 '<Entry><Employee>'
        UPDATETEXT #Work.XML @@ptrval NULL 0 @Name
        UPDATETEXT #Work.XML @@ptrval NULL 0 '</Employee><Number>'
        UPDATETEXT #Work.XML @@ptrval NULL 0 @Number
        UPDATETEXT #Work.XML @@ptrval NULL 0 '</Number><Type>'
        UPDATETEXT #Work.XML @@ptrval NULL 0 @Desc
        UPDATETEXT #Work.XML @@ptrval NULL 0 '</Type></Entry>'

        -- Переходим к следующей записи в курсоре.
        FETCH NEXT FROM WorkCursor INTO @Name, @Number, @Desc

    END

    -- Освобождаем курсор.
    DEALLOCATE WorkCursor

    -- Добавляем замыкающий элемент.
    UPDATETEXT #Work.XML @@ptrval NULL 0 '</EmployeeList>'

    -- Возвращаем набор результатов.
    SELECT * FROM#Work

END

GO
```

На рис. 9.10 показан результат исполнения хранимой процедуры `GetXML`.

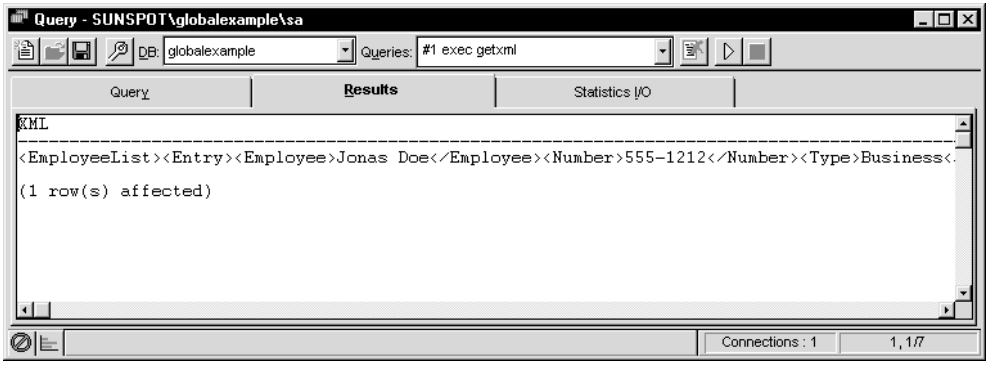


Рис. 9.10. Результат исполнения хранимой процедуры GetXML

Мы вернемся к этой хранимой процедуре, когда будем обсуждать код страницы ASP, который возвращает результаты такого запроса в необработанном виде. В следующем разделе будет рассмотрено, как, используя хранимую процедуру и набор данных ADO (ActiveX Data Objects) в странице ASP, возвратить XML-данные запросившему их клиенту. Последняя может принять любой набор записей ADO и возвратить из него XML-данные.

Создание XML-данных в промежуточных системах

До этого раздела мы обсуждали создание XML-данных непосредственно на SQL-сервере. Оставшаяся часть главы сконцентрирована вокруг среднего уровня (MiddleTier), как механизма возврата XML-данных и обновления SQL-сервера данными из поступивших XML записей.

На рис. 9.11 показаны результаты действия страницы `MiddleTierViewPhoneList.asp`. Хотя на экране присутствуют две одинаковые таблицы, сформированы они были различными способами. Верхняя таблица была создана при помощи страницы ASP `PhoneListUsingGetXML.asp`, которая применила описанную ранее хранимую процедуру `GetXML` в качестве исходного XML. Для создания нижней таблицы использовалась страница `PhoneListUsingXmlRecordSet.asp`, которая формирует XML из любого набора записей ADO. При обработке обеих страниц Active Server Pages использовался компонент ActiveX `XmlForAsp`, который является элементом управления, выполняющим циклическое обращение к анализатору MSXML. Поступать подобным образом приходится потому, что анализатор MSXML допускает на входе только URL (и осуществляет проверку безопасности) и не имеет средств для возврата вызывающему приложению строки XML-данных. Данный компонент может быть найден на сайте Microsoft (http://www.microsoft.com/xml) или на http://www.15seconds.com – ресурсе разработчиков страниц Active Server Pages.

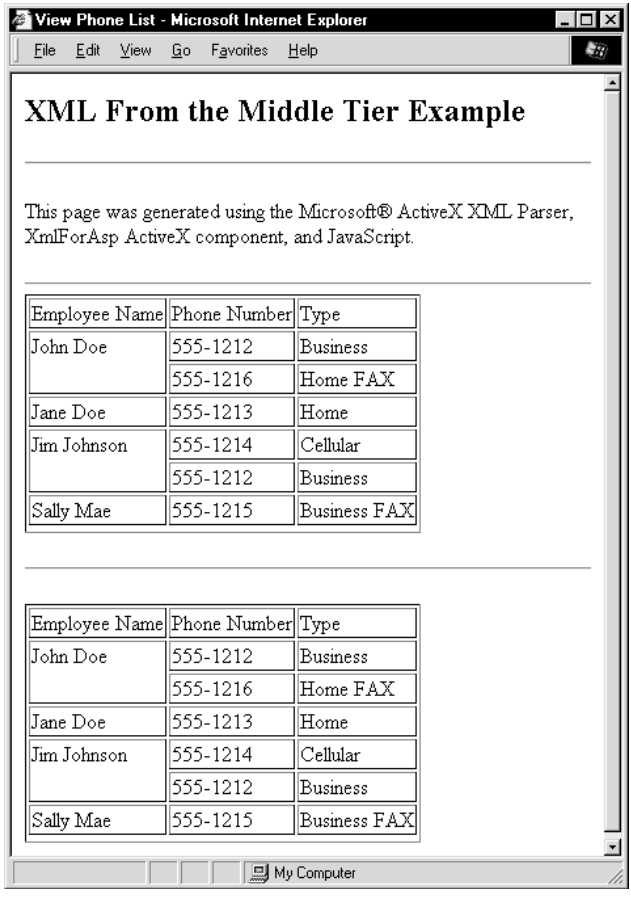
Страница, приведенная на рис. 9.11, сформирована тремя отдельными частями кода, запускаемого на сервере. Для начала рассмотрим код, возвращающий XML-данные для верхней таблицы, `PhoneListUsingGetXML.asp`.

Страница PhoneListUsingGetXML.asp

Этот небольшой ASP-файл создает связь с базой данных и вызывает хранимую процедуру `GetXML`, которая уже была изучена. Далее он выводит XML заголовок, `<?xml version="1.0"?>`, и за ним столбец XML из набора записей. Все, что делает этот файл, для клиента служит промежуточным звеном. Данные, которые он возвращает из запроса, обрабатываются на Web-сервере, превращающем их в HTML-таблицу (об этом будет сказано чуть позже). Ниже приведен код VBScript для этой страницы; полностью код может быть загружен с нашего Web-сайта http://webdev.wrox.co.uk/books/1525.

```
<!-- #include virtual="/XMLPhoneList/Connect.inc" -->

<%
    ' Создаем объект связи с базой данных.
    Set Conn = Server.CreateObject("ADODB.Connection")
    ' Сообщаем ему, какой источник данных ODBC использовать.
    Conn.Open strConnect
    ' Вызываем хранимую процедуру.
    Set XmlSet = Conn.Execute("{Call GetXML}")
    ' Добавляем заголовок XML, а затем одну строку/столбец.
%>
<?xml version "1.0"?>
<%=XmlSet("XML")%>
```



Employee Name	Phone Number	Type
John Doe	555-1212	Business
	555-1216	Home FAX
Jane Doe	555-1213	Home
Jim Johnson	555-1214	Cellular
	555-1212	Business
Sally Mae	555-1215	Business FAX

Employee Name	Phone Number	Type
John Doe	555-1212	Business
	555-1216	Home FAX
Jane Doe	555-1213	Home
Jim Johnson	555-1214	Cellular
	555-1212	Business
Sally Mae	555-1215	Business FAX

Рис. 9.11. Таблицы, сформированные при помощи страницы ASP

Страница PhoneListUsingXmlRecordSet.asp

Нижняя таблица, сформированная страницей `MiddleTierViewPhoneList.asp`, создана из другого XML-источника. Она использует в качестве поставщика XML-данных страницу `PhoneListUsingXmlRecordSet.asp`. Эта страница ASP использует команду `Include` на сервере, чтобы вставить групповую функцию для возврата XML-данных из набора записей, о чем мы расскажем в следующем разделе. Как и в предыдущей странице ASP, основная задача этой страницы состоит в том, чтобы установить связь с базой данных и выполнить хранимую процедуру. Хранимая процедура та же, что была использована в коде модуля Web Task. Она просто возвращает из базы данных с присоединенными откорректированными таблицами имя сотрудника, номер телефона и тип телефона, упорядоченные по номеру (ID) каждого сотрудника для согласования выходных результатов. Если не было выбрано упорядочивание по ID-номерам, то записи в двух таблицах, которые создаются во время выполнения `MiddleTierViewPhoneList.asp`, могут быть, а могут

и не быть расположены в одинаковм порядке. Этот код возвращает XML-строку, которая была создана при помощи `XmlFromRecordSet()`.

Ниже дан код страницы `PhoneListUsingXmlRecordSet.asp`, помещенной на нашем Web-сайте: http://webdev.wrox.co.uk/books/1525.

```
<!-- #include virtual="/XMLPhoneList/Connect.inc" -->
<!-- #include virtual="/XMLPhoneList/XMLFromRecordSet.inc" -->

<%
    ' В данном примере используется "универсальный" генератор XML из набора записей.
    ' Создаем объект связи с базой данных.
    Set Conn = Server.CreateObject("ADODB.Connection")
    ' Сообщаем ему, какой источник данных ODBC использовать.
    Conn.Open strConnect
    ' Вызываем хранимую процедуру, использующую Web Task.
    Set XmlSet = Conn.Execute("{Call XMLWebTaskQuery}")
    ' Используем функцию XMLFromRecordSet для возврата строки XML.
    strXml = XmlFromRecordSet(XmlSet,"Set","Row")
    ' Теперь выводим заголовок и строку.
%>
<?xml version="1.0"?>
<%=strXml%>
```

Универсальная функция для создания XML из группы записей

Страница `PhoneListUsingXmlRecordSet.asp` использует команду `Include` на сервере, чтобы вставить производящую функцию для возврата XML-данных из любого набора записей. Учитывая, что это производящая функция, ей требуется передать два элемента: имя, которое будет использовано для корневого узла, и имя, которое будет использовано для каждой возвращаемой записи. В приведенном выше примере корневой узел назван `Set`, а в XML-данных он будет выглядеть как `<Set>`. Элемент XML, циклически обрабатывающий записи, назван `Row`; в XML-данных он будет выглядеть как `<Row>`. Элементы, находящиеся внутри элемента `<Row>`, получают названия в соответствии с наименованиями полей в группе записей, которые изменяются для каждого нового объекта типа группы записей.

Для построения XML-данных эта функция использует локальные переменные. Сначала создается корневой элемент `<Set>`. Затем осуществляется циклический перебор записей в группе и добавление элементов `<Row>` и `</Row>` в начале и в конце полей данных. И наконец обрабатывается совокупность полей и создаются данные типа:

```
<FieldName>FieldValue</FieldName>
```

при использовании `Field.Name` и `Field.Value`. Следует учесть, что у `Field.Value` «отрезаны» пробелы с правой стороны. Это сделано потому, что SQL-сервер дополняет данные типа `char` пробелами справа до конца поля. Поскольку мы имеем дело с групповой функцией, использующей оба типа данных (`char` и `varchar`), лучше обезопасить себя и убрать ненужные пробелы. Не имеет смысла передавать лишние данные, поэтому пробелы отсекаются. Как только будут добавлены все записи, к строке присоединяется закрывающий элемент `</Set>`, а затем строка возвращается. Ниже дан код универсальной функции, которая создает XML-данные из групп записей; она

может быть загружена с нашего Web-сайта: http://webdev.wrox. co.uk/books/1525.

```
<%
    ' Данная функция возвращает строку XML из любого набора записей.
        ' Построение строки XML осуществляется путем:
        '   добавления обрамления набора Set;
        '   добавления обрамления строки Row для каждой строки данных;
        '   добавлением по столбцам open/data/close для каждого столбца Column
            в строке Row;
        '   добавлением замыкающего тэга строки Row;
        '   добавлением замыкающего тэга набора Set.

    Function XmlFromRecordSet(RecordSet,strSetTag,strRowTag)
        strXml = ""
        ' Теперь добавляем обрамление Set.
        strXml = strXml & "<" & strSetTag & ">"
        ' Убеждаемся, что возвращено сколько-то строк.
        If Not RecordSet.EOF And Not RecordSet.BOF Then
            ' Цикл по строкам.
            Do While Not RecordSet.EOF
                ' Добавляем открывающий тэг Row.
                strXml = strXml & "<" & strRowTag & ">"
                ' Теперь проходим по всем полям, используя имя..
                ' ..поля в качестве имени тэга и значение..
                ' ..поля в качестве значения тэга.
                For Each Field In RecordSet.Fields
                    ' Добавляем открывающий тэг.
                    strXml = strXml & "<" & Field.Name & ">"
                    ' Добавляем значение тэга. Обрезаем значение справа..
                    ' ..поскольку тип char() дополнен пробелами.
                    strXml = strXml & Rtrim(Field.Value)
                    ' Добавляем замыкающий тэг.
                    strXml = strXml & "</" & Field.Name & ">"
                Next
                ' Добавляем замыкающий тэг Row.
                strXml = strXml & "</" & strRowTag & ">"
                ' Переходим к следующей строке в наборе записей.
                RecordSet.MoveNext
            Loop
        End If
        ' Теперь добавляем замыкающий тэг Set
        strml = strXml + "</" & strSetTag & ">"
        ' Возвращаем строку вызывающей программе.
        XmlFromRecordSet = strXml
    End Function
%>
```

В табл. 9.1 сравниваются XML-данные, возвращенные двумя страницами Active Server Pages, так что налицо разница между групповым методом и вполнением хранимой процедуры `GetXML`.

Таблица 9.1. Сравнение группового метода и процедуры GetXML

PhoneListUsingGetXML.asp	PhoneListUsingXmlRecordSet.asp
<?xml version="1.0"?> <EmployeeList> <Entry> <Employee>Jonathon Doe </Employee> <Number>555-1212</Number> <Type>Business</Type> </Entry> ... </EmployeeList>	<?xml version="1.0"?> <Set> <Row> <Name>Jonathon Doe</Name> <Number>555-1212</Number> <Description>Business </Description> </Row> ... </Set>

Преобразование данных из XML в HTML-таблицы

Теперь наступил момент, когда данные должны быть преобразованы из XML в HTML-таблицы, показанные на рис. 9.11, на котором изображен код `MiddleTierViewPhoneList.asp` (в самом начале данного раздела). Код для построения таблиц, в сущности, совпадает с кодом `EmployeePhoneList.htm`, но для этого примера он был обобщен (см. полный код для `MiddleTierViewPhoneList.asp`). Сначала создается анализатор MSXML, затем вспомогательный компонент `XmlForAsp`. Далее очищаются строки, используемые для хранения описания таблицы. На следующем шаге загружается первый XML-документ с использованием помощника `XmlForAsp` для метода объекта `ReadFromFile()`, который принимает либо URL, либо имя файла; он считывает документ, расположенный по указанному адресу, и вставляет его в анализатор MSXML. Хотя анализатор MSXML тоже принимает URL в качестве способа загрузки данных, работа по этому способу в странице Active Server Pages приведет к возникновению ошибки защиты. Ошибка возникает потому, что фирма Microsoft разрабатывала анализатор MSXML для работы со стороны клиента, а не сервера; это означает, что защита должна гарантировать, чтобы данные пришли оттуда же, откуда получена страница. Объект `XmlForAsp` обходит это защитное ограничение.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">

<HTML>
<HEAD>
   <TITLE>View Phone List</TITLE>
</HEAD>

<SCRIPT LANGUAGE=JavaScript RUNAT=Server>

   // Создаем экземпляр XML-анализатора.
   var XmlDoc = new ActiveXObject("msxml");
   // XmlForScripting, вспомогательный
   var objXmlHelp = new ActiveXObject("XmlForAsp.XmlForScripting");
   // Строки для размещения строящейся таблицы.
   var strGetXmlTable = "";
   var strRecordSetTable = "";
```

```
// Загружаем документ по url хранимой процедуры.
objXmlHelp.ReadFromFile("http://" + Request.ServerVariables("SERVER_NAME") +
    "/XMLPhoneList/PhoneListUsingGetXML.asp",XmlDoc);
// Используем вариант с набором записей..
// ..и соответственно строим таблицу.
strGetXmlTable = BuildXmlTable("Employee","Number","Type");

// Загружаем документ из url набора записей.
objXmlHelp.ReadFromFile("http://" + Request.ServerVariables("SERVER_NAME") +
    "/XMLPhoneList/PhoneListUsingXmlRecordSet.asp",XmlDoc);

// Используем вариант с набором записей..
// ..и соответственно строим таблицу.
strRecordSetTable = BuildXmlTable("Name,"Number","Description");
```

Когда файл загружен, вызывается функция `BuildXmlTable()`. Следует учесть, что в данный момент она принимает три имени элементов, чтобы использовать их для построения HTML-таблицы (см. в приведенных выше сравнительных таблицах). Кроме того, функция возвращает строку, которая в этом коде выводится позднее. Затем аналогичная обработка осуществляется со вторым URL и создается вторая XML-строка. Далее обе эти строки выводятся в том же месте, где в предыдущем примере, `EmployeePhoneList.htm`, находился тэг `<SPAN>`.

Здесь приведен полный код `MiddleTierViewPhoneList.asp`, который может быть загружен с нашего Web-сайта: http://webdev.wrox.co.uk/books/1525.

```
function BuildXmlTable(strColName1,strColName2,strColName3)
{
    // Начинаем таблицу.
    var tableHTML = "<TABLE BORDER BORDERCOLOR=#000000><TR><TD>Employee
     _Name</TD><TD>Phone Number</TD><TD>Type</TD></TR>";
    // Получаем корень дерева XML.
    var root = XmlDoc.root;
    var nIndex = 0;
    // Получаем массив дочерних пунктов.
    var arrItems = root.children;
    // Текущая запись в телефонном списке.
    var entry = null;
    var nRowSpan = 0;
    // Проходим по всем строкам набора и добавляем их к таблице.
    for( nIndex = 0; nIndex < arrItems.length; nIndex++ )
    {
        // Получаем объект-строку по данному указателю.
        entry = arrItems.item(nIndex);
        if( nRowSpan == 0 )
        {
            // Вычисляем количество строк для счетчика.
            nRowSpan = CalcRowSpan(nIndex,arrItems,strColName1);
            // Теперь строим код html для данной записи.
            tableHTML = tableHTML + "<TR><TD VALIGN=TOP rowspan=" + nRowSpan + ">";
            tableHTML = tableHTML + entry.children.item(strColName1).text +
"</TD>";
        }
```

```
            tableHTML = tableHTML + "<TD>" + entry.children.item(strColName2).text +
                _"</TD>";
            tableHTML = tableHTML + "<TD>" + entry.children.item(strColName3).text
+
                _"</TD>";
            // Закрываем эту строку таблицы.
            tableHTML = tableHTML + "</TR>";
            // Уменьшаем счетчик строк, так чтобы он в конечном счете обнулился.
            nRowSpan--;
        }
        // Закрываем тэг таблицы.
        tableHTML = tableHTML + "</TABLE>"
        return tableHTML;
    }

    function CalcRowSpan(nStartIndex,arrEntries,strColName1)
    {
        // Сохраняем имя сотрудника.
        var strName = arrEntries.item(nStartIndex).children.item(strColName1).
text;
        // Разгружаем переменные контейнера.
        var strCurrName = strName;
        var nCount = 0;
        var nIndex = nStartIndex + 1;
        // Циклом проходим по массиву, подсчитывая вхождения имени сотрудника.
        while( nIndex < arrEntries.length && strName == strCurrName )
        {
            // Получаем имя следующего сотрудника.
            strCurrName = arrEntries.item(nIndex).children.item(strColName1).text;
            // Увеличиваем указатель и счетчик строк.
            nIndex++;
            nCount++;
        }
        return nCount;
    }

</SCRIPT>

<BODY BGPROPERTIES="FIXED" BGCOLOR="#FFFFFF">

<H2>XML From the Middle Tier Example</H2>

<HR>

<P>
This page was generated using the Microsoft&reg; ActiveX XML Parser, <BR>XmlForAsp
ActiveX component, and JavaScript.<BR>
</P>

<HR>

<%=strGetXmlTable%>

<BR>
<HR>
```

```
<BR>

<%=strRecordSetTable%>

</BODY>
</HTML>
```

Таким образом, мы создали XML при помощи SQL-сервера, вывели его на экран, используя HTML и JavaScript со стороны клиента, и осуществили обработку данных на среднем уровне (Middle Tier); при этом удалось избежать необходимости иметь анализатор MSXML на компьютере клиента. Последняя остановка нашего путешествия будет посвящена вопросу, как доставить XML на Web-сервер и обновить таким образом базу данных на стороне сервера.

HTML-форма для обновления списка телефонов

Все примеры до настоящего времени предназначались «только для чтения», что вполне удобно при создании отчетов. Однако большинство приложений клиент-сервер обладают некоторой формой обновления, добавления, изменения существующих данных. Оперативный торговый протокол (Online Trading Protocol, OTP), являющийся спецификацией деловых транзакций на основе XML, – это описание протокола, по которому сообщения в виде XML передаются и принимаются между двумя или более серверами и, возможно, клиентом. Спецификация открытого финансового обмена (Open Financial eXchange, OFX), пока еще не базирующаяся на XML, продвигается в этом же направлении, сближаясь с OFX/Gold, известным также под названием IFX. OFX концентрируется на доставке финансовых транзакций от финансовых менеджеров (Quicken, Money) в банки и маклерам. Обе эти реализации протокола на сервере преобразуют XML-запросы в исполнение на другом конце существующей системы.

В качестве примера того, как следует принимать XML-запросы, приводится последнее описанное в этой главе мини-приложение, которое принимает XML-команды от клиента и обновляет базу данных. Оно используется для обновления одного имени, одного номера телефона, одного описания телефона или любой комбинации этих элементов. Изменения производятся при помощи HTML-формы, создающей XML-строку. Эта строка посылается (`POST`) в страницу ASP, которая анализирует XML-запрос и в соответствии с ним обновляет базу данных. На рис. 9.12 показано, как выглядит HTML-страница до обновления.

А рис. 9.13 представляет ту же страницу после обновления.

Чтобы изменить данные, следует ввести из таблицы, показанной внизу, прежнее имя, которое требуется изменить, и за ним новое имя, в таком виде как оно должно выглядеть. Аналогично следует поступить с номером и типом телефона. Когда поля заполнены, следует щелкнуть по кнопке **Build XML**, которая создает из полей XML-строку и вводит ее в поле **XML Text** (XML-текст).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">

<HTML>
<HEAD>
   <TITLE>Update Phone List Entry Form</TITLE>
</HEAD>
```

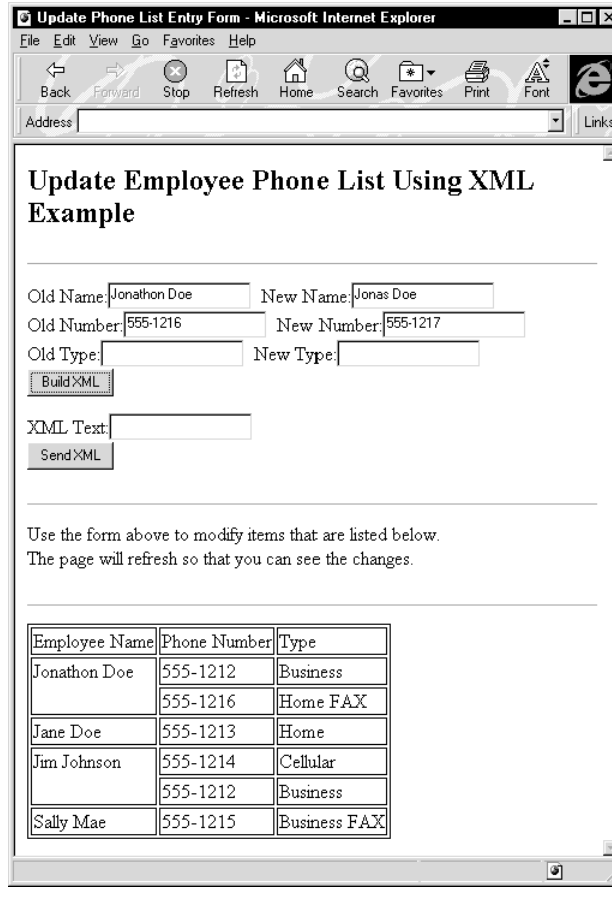


Рис. 9.12 HTML-страница до обновления базы данных

```
<SCRIPT LANGUAGE="JavaScript">
   //<!--
   function OnBuildXML()
   {
       // Строим строку, содержащую XML, необходимый для обновления базы данных.
       UpdateForm.XmlString.value = "<UpdateList><UpdateEntry><OldName>" +
           BuildForm.OldName.value + "</OldName><NewName>" +
           BuildForm.NewName.value + "</NewName><OldNumber>" +
           BuildForm.OldNumber.value + "</OldNumber><NewNumber>" +
           BuildForm.NewNumber.value + "</NewNumber><OldType>" +
           BuildForm.OldType.vaue + "</OldType><NewType>" +
           BuildForm.NewType.vlue + "</NewType></UpdateEntry></UpdateList>";
   }
   //-->
</SCRIPT>
```

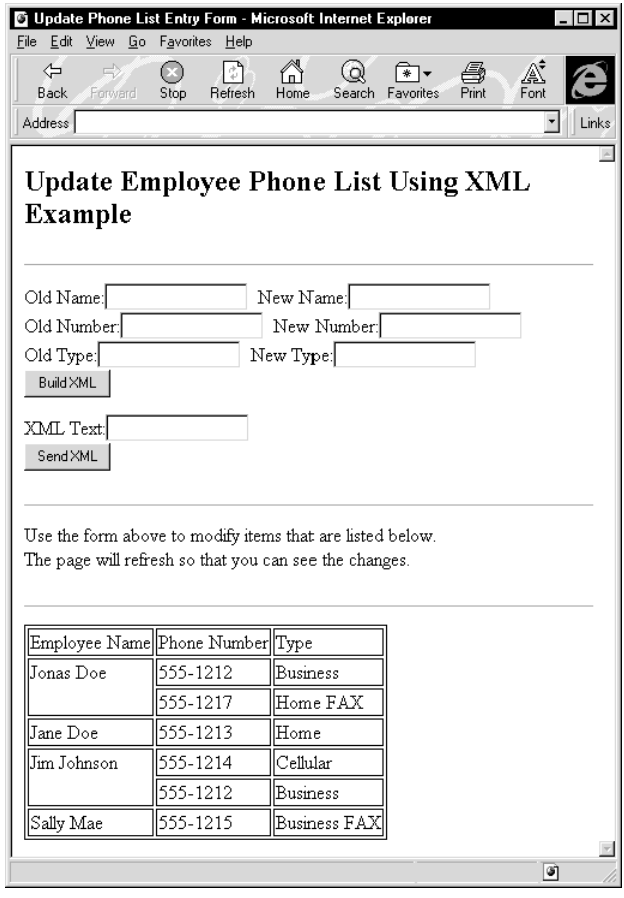


Рис. 9.13 HTML-страница после обновления базы данных

Для отсылки XML-строки на Web-сервер щелкните по кнопке **Send XML**, которая отправит (`POST`) данные на страницу `UptadePhoneList.asp`.

Чтобы увидеть, как это осуществляется, рассмотрим исходный код страницы ASP, который формирует HTML-страницу.

```
<BODY BGPROPERTIES="FIXED" BGCOLOR="#FFFFDD">

<H2>Update Employee Phone List Using XML Example</H2>

<HR>

<FORM METHOD="post" NAME="BuildForm">
Old Name:<input type="Text" name="OldName" value="">
New Name:<input type="Text" name="NewName" value=""><br>
Old Number:<input type="Text" name="OldNumber" value="">
New Number:<input type="Text" name="NewNumber" value=""><br>
Old Type:<input type="Text" name="OldType" value="">
New Type:<input type="Text" name="NewType" value=""><br>
```

```
INPUT TYPE="button" VALUE="Build XML" NAME="BuildXml" ONCLICK="OnBuildXml()">
</FORM>

<form method="post" name="UpdateForm"_
action="<%="http://" & Request.ServerVariables("SERVER_NAME") & _
"/XMLPhoneList/UpdatePhoneList.asp"%>">
XML Text:<input type="Text" name="XmlString" value=""><br>
<input type="submit" value="Send XML" name="Send XML"></p>
</form>
<HR>

<P>
Use the form above to modify items that are listed below.<br>
The page will refresh so that you can see the changes.
</P>

<HR>
```

Эта страница – простой участок HTML-кода, созданный при помощи ASP и состоящий из двух форм и таблицы. Код таблицы построен с использованием предыдущих примеров, но с небольшими изменениями. Клиенту возвращаются две формы (см. выше): одна с пользовательским методом обработки событий `onClick` (по нажатию), который формирует XML-строку и копирует ее в поле **XML Text** (XML-текст); другая со свойством *действие* (action), указывающим на URL

http://webdev.wrox.co.uk/books/1525XMLPhoneList/UpdatePhoneList.asp.

Реальная работа выполняется на Web-сервере. Внизу приведен код для страницы Active Server Page, которая создает HTML-страницу **Update Phone List Entry**. Полный код может быть загружен с нашего Web-сайта: http://webdev.wrox.co.uk/books/1525.

```
<SCRIPT LANGUAGE=Javacript RUNAT=Server>

   // Это код ASP для загрузки данных XML из файла..
   // ..и обработки файла на сервере.

   // Контейнер Xml-документа.
   var XmlDoc = new ActiveXObject("msxml");
   // XmlForScripting, вспомогательный
   var objXmlHelp = new ActiveXObject("XmlForAsp.XmlForScripting");
   // Загружаем из URL
   objXmlHelp.ReadFromFile("http://" + Request.ServerVariables("SERVER_NAME") +
       "/XMLPhoneList/PhoneListUsingGetXML.asp",XmlDoc);

   function CalcRowSpan(nStartIndex,arrEntries)
   {
       // Сохраняем имя сотрудника.
       var strName =
arrEntries.item(nStartIndex).children.item("Employee").text;
       // Разгружаем переменные контейнера.
       var strCurrName = strName;
       var nCount = 0;
       var nIndex = nStartIndex + 1;
       // Проходим циклом по массиву, подсчитывая число вхождений имени
сотрудника.
```

```
        while( nIndex < arrEntries.length && strName == strCurrName )
        {
            // Получаем имя следующего сотрудника.
            strCurrName = arrEntries.item(nIndex).children.item("Employee").text;
            // Увеличиваем указатель и счетчик строк.
            nIndex++;
            nCount++;
        }
        return nCount;
    }
    // Начинаем таблицу.
    var tableHTML = "<TABLE BORDER BORDERCOLOR=#000000><TR><TD>Employee
        Name</TD><TD>Phone Number</TD><TD>Type</TD></TR>";
    // Получаем корень дерева XML <ITEMLIST>
        var root = XmlDoc.root;
    var nIndex = 0;
    // Получаем массив дочерних пунктов.
    var arrItems = root.children;
    // Это текущая запись в телефонном списке.
    var entry = null;
    var nRowSpan = 0;
    // Проходим циклом по всем элементам <Entry> в списке <EmployeeList>
    // и добавляем их к таблице.
    for( nIndex = 0; nIndex < arrItems.length; nIndex++ )
    {
        // Получаем объект <Entry> по данному указателю.
        entry = arrItems.item(nIndex);
        if( nRowSpan == 0 )
        {
            // Подсчитываем число строк для счетчика.
            nRowSpan = CalcRowSpan(nIndex,arrItems);
            // Теперь стоим код html для данной записи.
            tableHTML = tableHTML + "<TR><TD VALIGN=TOP ROWSPAN=" + nRowSpan +
">";
            tableHTML = tableHTML + entry.children.item("Employee").text +
"</TD>";
        }
        tableHTML = tableHTML + "<TD>" + entry.children.item("Number").text +
"</TD>";
        tableHTML = tableHTML + "<TD>" + entry.children.item("Type").text + "</
TD>";
        // Close this line of the table
        tableHTML = tableHTML + "</TR>";
        // Уменьшаем счетчик строк, так чтобы он в конечном счете обнулился.
        nRowSpan--;
    }
    // Закрываем тэг таблицы.
    tableHTML = tableHTML + "</TABLE>"
</SCRIPT>
```

```
<%=tableHTML%>

</BODY>
</HTML>
```

А вот XML-строка, созданная `OnBuildXml()`:

```
<UpdateList>
   <UpdateEntry>
       <OldName>John Doe</OldName>
       <NewName>John Doe Sr.</NewName>
       <OldNumber></OldNumber>
       <NewNumber></NewNumber>
       <OldType>Fax machine</OldType>
       <NewType>Dial up modem</NewType>
   </UpdateEntry>
</UpdateList>
```

Когда XML-данные приходят на Web-сервер (в форме данных, которые мы видели в предыдущем отрывке кода для вывода XML), сервер принимает XML-строку из объекта запроса и использует компонент ActiveX помощника `XmlForAsp` для ввода строки в анализатор MSXML (показанный в следующем разделе кода). Это дает такой же эффект, как если бы документ загружался с URL: анализатор конвертирует XML-данные в иерархию объектов.

```
<!--#include virtual="/XMLPhoneList/Connect.inc"-->

<%
    ' Обновляет запись в списке телефонов с использованием XML.

    ' Требует создания экземпляра XML-анализатора.
    Set objXmlDoc = CreateObject("msxml")
    ' Создает экземпляр вспомогательной ASP для MSXML.
    Set objXmlHelp = CreateObject("XmlForAsp.XmlForScripting")
    ' Вставляет все, что передано в XML-строке в анализатор.
    objXmlHelp.XmlString(objXmlDoc) = Request("XmlString")
```

После того как данные переведены в форму объектов, создается связь с базой данных. Далее код отыскивает корневой узел и его дочерние узлы, которые могут содержать нуль или более элементов `<UpdateEntry>...</UpdateEntry>`. Для каждого дочернего элемента из коллекции элементов создается и выполняется серия из трех SQL-инструкций. Каждая из инструкций обновляет базу данных, используя старое значение в качестве ключа для нахождения записи, подлежащей обновлению. Ниже приведен код страницы Active Sever Page для обработки запроса на обновление. Полностью код может быть загружен с нашего Web-сайта: http://webdev.wrox.co.uk/books/1525.

```
    ' Создает объект связи с базой данных.
    Set Conn = Server.CreateObject("ADODB.Connection")
    ' Сообщает ему, какой источник данных ODBC следует использовать.
    Conn.Open strConnect

    ' Получает корень дерева XML.
    Set root = objXmlDoc.root
```

```
    nIndex = 0
    ' Получает массив дочерних пунктов.
    Set arrItems = root.children
    ' Проходит циклом по всем запросам на обновление и обновляет базу данных.
    For nIndex = 0 To arrItems.length - 1
        ' Получает объект обновления по данному указателю.
        Set objUpdate = arrItems.item(nIndex)
        ' Строит строку SQL для обновления имени.
        strSql = "UPDATE Employee set Name = '" & _
            objUpdate.children.item("NewName").text & "' where Name = '" & _
            objUpdate.children.item("OldName").text & "'"
        ' Теперь выполняет обновление.
        Conn.Execute(strSql)
        ' Строит строку SQL для обновления номера телефона.
        strSql = "UPDATE Phone set Number = '" & _
            objUpdate.children.item("NewNumber").text & "' where Number = "'_
                & objUpdate.children.item("OldNumber").text & "'"
        ' Теперь выполняет обновление.
        Conn.Execute(strSql)
        ' Строит строку SQL для обновления типа телефона.
        strSql = "UPDATE PhoneType set Description = '" & _
            objUpdate.children.item("NewType").text & "' where _
                Description = '" & objUpdate.children.item("OldType").text & "'"
        ' Теперь выполняет обновление.
        Conn.Execute(strSql)
    Next
%>
```

Этот пример может быть использован для проведения любого количества обновлений. HTML-страница, показанная здесь, посылает на выполнение только по одному элементу `<UpdateEntry>...</UpdateEntry>` за раз. Однако сервер отдела может сгруппировывать изменения в один более крупный файл и вносить их в основной список раз в неделю. Аналогичный подход может быть использован для любого типа пакетных изменений. Эта концепция применима и для больших приложений, которые взаимодействуют с несколькими конечными *хостами* (host, головная машина) для выполнения запросов.

Заключение

В этой главе мы показали:

- как использовать SQL-сервер для создания XML-данных для отчетов;
- как отобразить эти данные в форме таблицы;
- как преобразовать XML-запросы в обновление SQL.

Подходы, примененные в этой главе, должны обеспечить специалистов базовыми инструментами, которые необходимы для построения сложных систем клиент-сервер. Используя хранимые процедуры, модули Web Task, обработку на среднем уровне (Middle Tier), читатели должны суметь распределить рабочую нагрузку между несколькими уровнями и компьютерными архитектурами, получив в результате устойчивые системы с быстрой реакцией.

Глава 10. XML на стороне сервера

Обратите внимание *Перед запуском примеров приложений настоятельно советуем проверить, не появились ли на нашем Web-сайте изменения к данной главе. Мы предоставим изменения к данной главе, а также к файлам с примерами приложений на нашем Web-сайте: http://webdev.wrox.co.uk/books/1525/.*

XML является быстро развивающейся областью, прогресс в этом отношении был особенно заметен во время подготовки сборника. В то время как теория, описанная здесь, остается все той же, общедоступные компоненты анализаторов, которые были использованы в наших примерах, могли быть изменены в соответствии с изменениями стандартов XML. Например, названия свойств и методов вряд ли останутся такими же, какие были использованы в этой главе.

Во избежание ошибок и разочарований, прежде чем устанавливать и запускать приложение, проверьте, пожалуйста, нет ли на нашем Web-сайте дополнений к нему.

Многие специалисты порой склонны размышлять об XML-документах исключительно как о средстве разметки фиксированных документов и их анализа с целью получения информации. Однако XML можно рассматривать и как самоописывающийся формат данных, исполняемый исключительно в тексте. При таком понимании он становится идеальным механизмом для обмена информацией между клиентом и сервером. Любой анализ, который допускается осуществлять в автономной системе или Web-браузере, может выполняться и на сервере. Такое отношение к XML как общему языку Web открывает двери для новых возможностей в терминах клиентов и серверов. Появится возможность создавать серверы, которые не подразумевают, что клиент – это пользователь-человек, ожидающий увидеть содержание.

Когда мы говорим «сервер», мы, кроме того, подразумеваем клиента и сеть между ними. Приложения Internet и intranet обнаруживают некоторые возможности, для использования которых XML представляется идеальным выбором. Хотя существуют более изощренные программы отображения данных, текст XML допускает весьма свободную связь между клиентами и серверами. Будущие клиенты смогут выяснить у сервера желаемый формат обмена, который прежде был не известен клиенту, и осуществить транзакцию. Мы проиллюстрируем решение специальных задач на примере несложного intranet-приложения, которое позволя-

ет пользователю искать и находить технические статьи, размеченные в XML. Это приложение продемонстрирует принципы динамической компоновки и анализа XML на обеих сторонах: на сервере и на компьютере клиента.

В общем случае клиент-серверное XML-приложение должно решать ряд специфических и сложных задач:

- форматировать сообщения для сервера;
- пересылать XML-документы между клиентом и сервером;
- разбирать на сервере сообщения от клиента и выполнять поставленные задачи;
- динамически генерировать XML-документ и возвращать его клиенту.

Причины использования XML на сервере

Обычные приложения клиент-сервер тесно связаны между собой. Они совместно используют бинарные форматы сообщений и протоколы передачи данных. Даже если платформы клиента и сервера различны, например, при дистанционном вызове клиентом, работающим в среде Windows, сервера, использующего Unix, клиенту и серверу все равно предписан общий бинарный формат передачи данных. Программные продукты, установленные на сервере и на стороне клиента, как правило, должны исходить от одного разработчика или, по крайней мере, должны быть созданы с использованием одинакового инструментария. Такой подход годится для четко определенных приложений, работающих при централизованном управлении. Бинарные форматы намного компактнее, чем текст, хотя текст тоже хорошо сжимается. Организация разветвленной системы со строгой привязкой к явному API или протоколу, использующему бинарный формат для передачи данных, является самым эффективным решением. К сожалению, подобное построение системы накладывает серьезные ограничения на всех участников этого процесса, вот почему специализированные протоколы используются лишь небольшим числом групп разработчиков, а их программные продукты не получают достаточного признания. Проблема заключается в том, что бинарные протоколы требуют скрупулезного соблюдения правил. Они нетерпимы к малейшим ошибкам и практически не позволяют вносить дополнения без предварительного согласования. В то же самое время стандартные протоколы, такие как HTTP и FTP, распространились повсеместно. Трудно представить, чтобы сотни и тысячи других протоколов достигли такого же уровня популярности.

Теперь поговорим о World Wide Web. Авторы Web-сайтов редко являются разработчиками Web-браузеров. Между авторами и их «читателями», просматривающими Web-страницы, нет никакой координации. Тем не менее, функциональные возможности Web-сервера доступны любому браузеру, обладающему необходимыми свойствами. Базовый, стандартный HTML работает повсеместно, а оригинальные дополнения, скорее всего, несколько ухудшат данную страницу, но не испортят ее совсем. Все зависит от HTTP-протокола и HTML. Таким образом, для разработки Web-сайта не требуется ничего, кроме подходящего текстового редактора. Мы готовы, как бы негласно заявляет Internet, работать с любым клиентом и разговариваем на языке, понятном всем клиентам. При разработке

Internet-приложения необходимо обеспечить такую же степень свободы. Используя XML, можно предоставить наш сервер любому клиенту, который разделяет наш взгляд на решаемую задачу. Кладовщики и продавцы, доктора и пациенты – пока у нас у всех есть общие интересы, мы способны вести электронный диалог.

XML позволяет писать приложения, исключительно терпимые к мелким погрешностям, при условии, что мы не будем настаивать на строгой оценке состоятельности документа на основе его схемы. Если тэги соответствуют основным правилам XML, такой документ, как уже было сказано, называется правильным. В свою очередь, правильный документ, который выполнен при строгом соблюдении некоторой недвусмысленной схемы, называется состоятельным. Если предполагается, что документ соответствует некоторой схеме, но в данном случае нет необходимости в строгом ее соблюдении, можно адаптироваться к незначительным ошибкам в выполнении требований этой схемы. Элементы, заключенные в тэги, описывают себя сами. Стоит только взглянуть на имя тэга, и сразу становится ясно, о чем идет речь. Тэги могут располагаться в любом порядке, допускаемом схемой документа. В худшем случае пользователь-человек может рассмотреть код XML и даже разобраться в нем, что очень важно при разработке и отладке. В наилучшем случае XML отделит данные и их смысл от их визуального представления. Для осуществления контакта клиент-сервер ни клиенту, ни серверу не требуется знание того, что происходит на другом конце линии.

Поскольку мы таким образом освобождаемся от оков обязательного единого инструментария, следует ожидать резкого расширения возможных типов клиентов. В настоящее время мы предполагаем человека, работающего с браузером, и автоматизированный процесс на сервере. А что в будущем? Клиенты и серверы становятся все более многообразными и интересными.

Клиенты: агенты, браузеры и другие

Обычно Web-приложения типа клиент-сервер состоят из программы-пользователя, взаимодействующей с Web-браузером на стороне клиента, и некоторых процессов на сервере, генерирующих HTML. Обмен данными происходит синхронно. В системах подобного типа нет особых побуждений к использованию XML. В этом разделе будут рассмотрены некоторые операции на сервере, результат которых можно просто передавать в виде HTML и не применять обработку на стороне клиента. Но если вывести пользователя за пределы мгновенной транзакции, ситуация резко изменится. Можно представить себе, что на стороне клиента происходит некий автоматизированный процесс, например какое-то приложение осуществляет последовательную передачу данных, наподобие электронного обмена данными (Electronic Data Interchange, EDI), но только с применением XML вместо бинарных форматов. XML-документы не только легче записывать, чем форматы EDI, но вдобавок XML можно передавать посредством протокола HTTP. В то время как для разрешения прохода данных EDI необходима специальная конфигурация брандмауэров, настройка XML, как правило, допускает свободный проход потока HTTP. Итак, в общем случае XML-приложение может быть выполнено с меньшими усилиями и хлопотами, чем формальная реализация EDI-про-

граммы. Более подробная информация об использовании XML для EDI находится на http://www.xmledi.com.

Спецификация обмена данными в здравоохранении (называемая Health Level Seven) подразумевает использование языка SGML в качестве единственного формата кодирования, однако в будущем к нему может быть добавлен и XML. В этом случае требуется, чтобы данные оставались *данными*, а их преобразование в приятную для глаза форму откладывается до окончательного представления их пользователю.

Передача данных, если говорить в широком смысле, не обязана быть синхронной. Электронные приложения, в особенности для коммерции, исполнялись бы лучше при использовании надежной асинхронной системы сообщений. XML генерировался бы и выстраивался в очередь для передачи. Полученный XML мог бы некоторое время находится на сервере до осуществления обработки или передачи другому серверу для дополнительной обработки. XML-обмен может быть частью длинной транзакции, в которой предполагается, что запрограммированный агент будет уведомлен один раз при получении XML-данных и еще раз, когда транзакция окончится. Теперь не обязательно создавать приложения для сервера на основе только синхронных моделей. Мы в какой-то мере освободились от браузеров и ввели новые типы клиентов, такие как e-mail и автоматизированные процессы на стороне клиента, реагирующие на сообщения или действия баз данных.

Много говорилось об автоматических агентах, странствующих по Internet и выполняющих задачи от имени своих пользователей. XML – великолепный формат для их взаимодействия. Он прост и легко реализуется на любой платформе. В силу того, что XML использует тэги, агентам достаточно, чтобы XML-документ был правильным, и они могут быть весьма терпимы к ошибкам, игнорируя тэги, которых не понимают. Если, например, после того как была реализована версия принимающего агента, описание DTD пополнилось новыми тэгами, или – другая ситуация – программист-разработчик неверно понимал DTD, то документ будет несостоятельным. Когда агент обнаружит несостоятельный XML-документ, он может понизить уровень требований и обрабатывать данные как правильные. Когда спецификация пространств имен (описанная в главе 4) будет завершена, агенты смогут получать по Internet обновленные схемы от разработчика DTD.

Современное понимание термина «клиент», по-видимому, будет расширяться по мере увеличения числа областей, где используются сетевые технологии. XML – простая и легко внедряемая схема форматов, подходящая для слабо связанных, управляемых данными клиентов, с которыми мы познакомимся. Рассмотрим обычную систему, которая, используя XML, позволяет серверу работать с нетрадиционными клиентами.

Система хранения технических статей

Мы собираемся построить простую систему, обслуживающую, предположим, наш технический архив. Каждая статья хранится на сервере в XML-формате, использующем простое описание DTD. Статья содержит название, биографию автора и собственно текст (тело статьи). Кроме того, в тело статьи включена

аннотация. Имеется также база данных, содержащая ключевую информацию о каждой статье, например название, имя автора и дату публикации. Мы хотим, чтобы пользователь мог задать некий критерий поиска и получить таблицу со статей, удовлетворяющей заданному критерию. Выбор статьи приводит к получению от сервера XML-файла и его визуальному представлению в окне браузера с использованием языка XSL для выполнения форматирования в соответствии с правилами стиля, находящимися на сервере.

Пример можно запустить непосредственно с нашего Web-сайта, или загрузить код и построить его самостоятельно. Оба варианта доступны на http://webdev.wrox.co.uk/books/1525.

Уточнение *Для этого приложения требуется файл msxml.dll из бета-версии IE 5.0 beta (к моменту написания книги версия 5.0 IE еще не была реализована). Получив этот файл, запустите IE4 и в командной строке наберите "regsvr32 msxml.dll". Таким образом, вы зарегистрируете новый XML-анализатор, заменив им старый (IE4) анализатор. Файл msxml.dll устанавливается в директорию Winnt\system32. Можно установить новый анализатор в другую директорию, с тем чтобы позднее вернуться к старой версии, перерегистрировав ее.*

Клиент

Пока ничего нового для подготовленного читателя сказано не было. Всем нам приходилось наблюдать, а то и строить системы, которые посылали данные из HTML-форм на сервер и получали динамически сформированные HTML-данные в качестве ответа. К сожалению, подобная схема ограничивает круг возможных клиентов Web-браузерами. Представление только для просмотра исключает предполагаемых клиентов, которым необходимы только данные. Мы собираемся использовать XML для достижения двух целей: сохранять различия между самими данными и формой их представления, а также обмениваться информацией с сервером, применяя способ, устойчивый к изменениям и ошибкам в формате обмениваемых данных. Наш клиент будет создавать XML-данные, представляющие параметры поиска, а затем анализировать XML-данные от сервера, содержащие результаты поиска. При подобном обмене XML-данными, динамически созданными на месте, наш сервер, помимо графического клиента, которого мы построим для иллюстрации идеи, будет полезен для самых разнообразных клиентов.

Создание запросов

Начнем традиционно. Первая страница нашего клиента (рис. 10.1) содержит элементы HTML-формы для занесения критериев поиска в базе данных по автору, названию и дате публикации. Не показаны компонент XML-анализатора и пустая область `<DIV>`. Кроме того, на странице размещены несколько функций JavaScript для обеспечения ряда функциональных возможностй на стороне клиента.

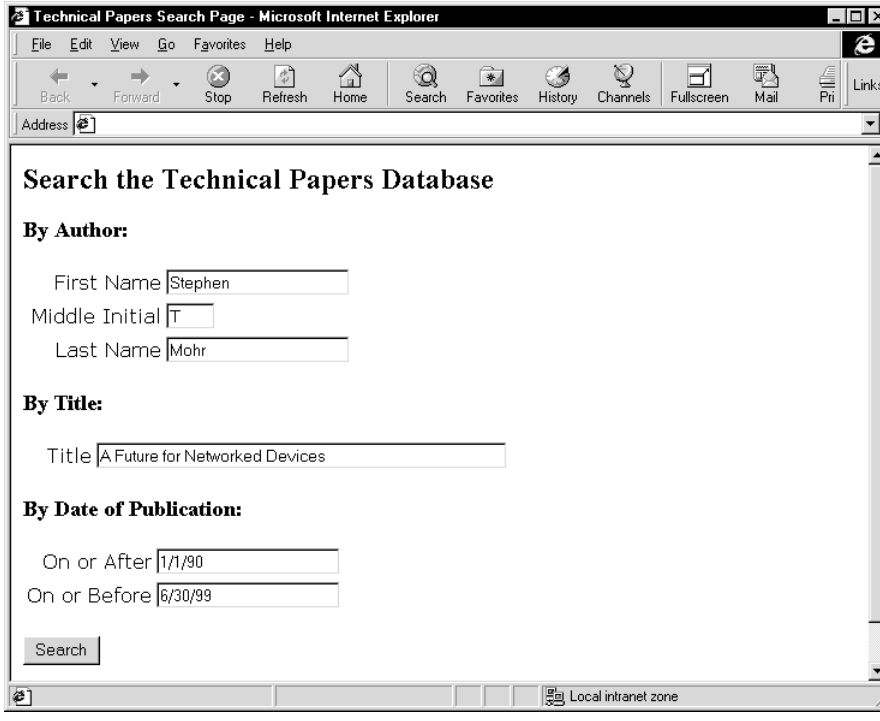


Рис. 10.1. Форма для ввода запроса, расположенная на странице клиента

Совет *База данных, используемая в этом примере, расположена, как и остальной код, на странице http://webdev.wrox.co.uk/books/1525. Она поддерживает дату публикации в виде поля типа date. Можно вводить дату в любом формате короткой даты (short-date format).*

Функции JavaScript будут использоваться, как это реализовано в Microsoft Internet Explorer версии 4.x, для выделения критериев и форматирования запроса на сервер в виде XML. Мы остановились на этом выборе инструментов потому, что собираемся использовать приспособленный для работы со скриптами XML-анализатор, написанный в виде компонента COM. Если вы хотите попробовать выполнить пример с браузерами Netscape, вам потребуется дополнительный компонент (plug-in), чтобы использовать такие компоненты Active X, как анализатор. К тому же исполнение DHTML в Netscape отличается от того, что реализовано в Microsoft, так что вам, возможно, потребуется в нескольких местах адаптировать код скриптов. Для упрощения мы будем работать только с Internet Explorer. Самое существенное, что ответ будет вставляться в область `<DIV>` внизу страницы, так что пользователю не придется покидать страницу поиска для просмотра результатов вне зависимости от того, сколько поисков он проводит.

Совет *Во время подготовки сборника ActiveX XML-анализатор претерпел значительные изменения. Новая версия, находящаяся на этапе бета-тестирования и запланированная к поставке вместе с Internet Explorer 5.0, предлагает новую объектную модель, которая следует развиваемой консорциумом W3C объектной модели XML-документа (DOM). Я использовал именно эту версию компонента XML-анализатора, а не предыдущий COM-анализатор фирмы Microsoft, чтобы проиллюстрировать новую модель DOM. К моменту завершения работ над DOM в интерфейсе анализатора могут появиться некоторые незначительные изменения. Основные различия в объектных моделях, предлагаемых двумя версиями, вы можете найти на http://www.microsoft.com/xml.*

Для простоты и ясности я ограничил этот пример использованием Microsoft DHTML. Безусловно, в реальном приложении необходимо будет определять тип браузера и возвращать разработанную под данный тип страницу поиска.

Получение списков статей

Результаты каждого поиска обнаруживаются в виде HTML-страницы (рис. 10.2). Название статьи появляется в виде гиперссылки на страницу, которая возвращает выбранную статью. Однако результаты поиска пересылаются с сервера в виде XML, так что приходится по мере разбора ответа динамически создавать таблицу на стороне клиента. Анализатор на странице поиска отыскивает и анализирует XML-данные, полученные от сервера. Когда результаты поиска проанализированы, функции JavaScript на странице поиска проходят по дереву разбора и создают данные HTML, которые динамически вставляются в область <DIV> внизу страницы.

Просмотр статей

Когда пользователь щелкает по гиперссылке, имя файла выбранной статьи передается в качестве параметра в страницу Active Server Page (ASP). При этом Web-страница динамически создается и посылается клиенту. Эта страница содержит компонент XSL-процессора, который обеспечивает данным полное визуальное представление. Компонент отыскивает статью и таблицу с соответствующими XSL-правилами. Он применяет эти правила к содержанию XML и создает HTML-код красиво оформленной страницы, которая показана на рис. 10.3.

Сервер

До сих пор мы не обсуждали, как работает сервер, хотя он по сути является сердцем всей системы. Основным элементом в этом случае служит страница ASP, сочетающая в себе компоненты JavaScript и ActiveX для анализа XML, осуществления запросов к базе данных, а затем динамического формированя XML для отправки обратно клиенту.

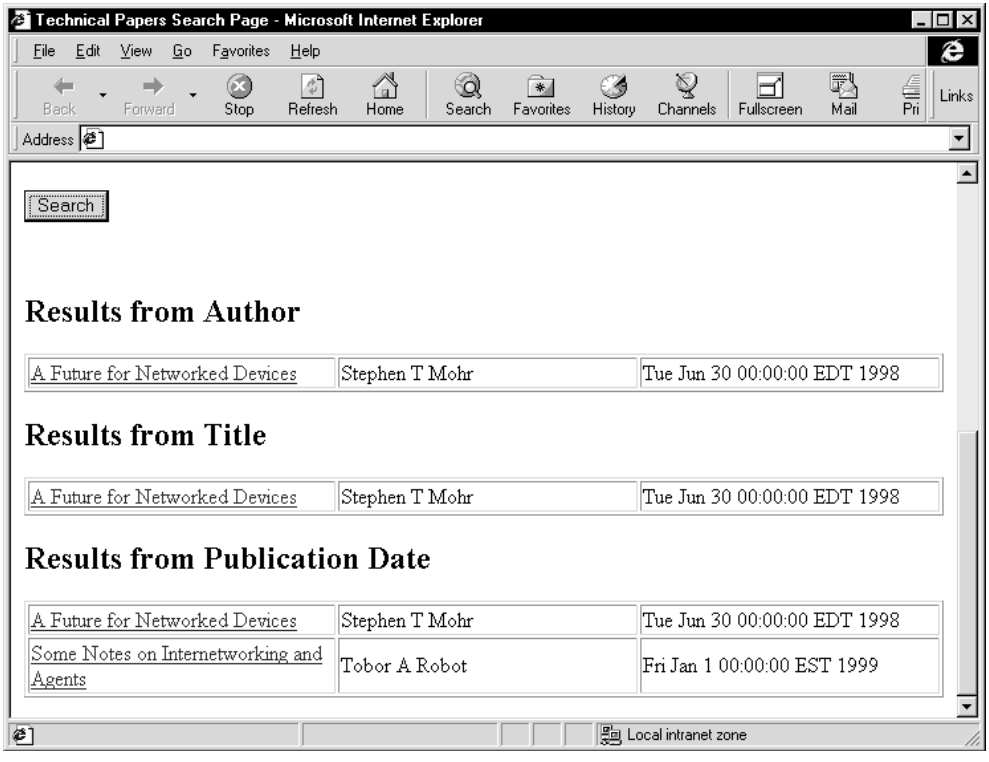


Рис. 10.2. HTML-страница с результатами поиска

Формирование запросов

XML-данные, посланные клиентом, анализируются на сервере таким же анализатором, какой использован на стороне клиента. Для обхода дерева разбора снова используется JavaScript. Однако на этот раз формируется и выполняется SQL-запрос для каждого поиска, затребованного клиентом. При возвращении каждого набора результатов производится его итеративная обработка, и XML-данные, содержащие результаты поиска, отправляются клиенту. В нашем примере будут использованы объекты ADO, набор компонентов COM, реализованный фирмой Microsoft для работы с базой данных, хотя для него подошли бы любые механизмы доступа к базе данных, которые можно запустить из страницы ASP. Ниже приведено определение типа документа (DTD) для запроса поиска статей.

```
<!ELEMENT PaperQuery (BYAUTHOR | BYTITLE | BYDATE)+>

<!ELEMENT BYAUTHOR (FNAME?, MI?, LNAME?)>
<!ELEMENT FNAME (#PCDATA)>
<!ELEMENT MI (#PCDATA)>
<!ELEMENT LNAME (#PCDATA)>

<!ELEMENT BYTITLE TITLE>
```

```
<!ELEMENT TITLE (#PCDATA)>

<!ELEMENT BYDATE (AFTER?, BEFORE?)>
<!ELEMEN AFTER (#PCDATA)>
<!ELEMENT BEFORE (#PCDATA)>
```

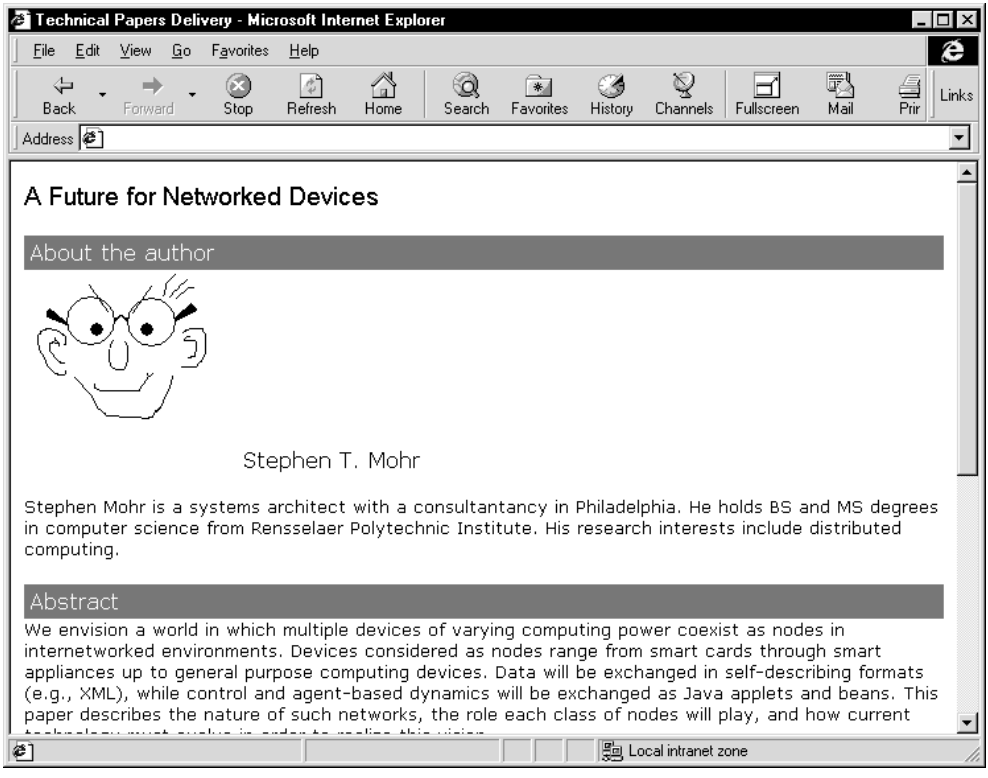


Рис. 10.3. Просмотр отформатированной статьи

Тэг <BYAUTHOR> (по автору) на первый взгляд может показаться странным: мы сделали необязательными все компоненты. Это одна из тех проблем, возникающих, когда желательные для клиента возможности трудно со всей точностью выразить на XML. Мы хотим, чтобы пользователи могли искать статьи, применяя любую комбинацию частей имени автора, при условии, что введена хотя бы какая-то часть имени. Это пожелание трудно, если вообще возможно, выразить в строгом синтаксисе DTD.

Тэг <BYTITLE> (по названию) вводит косвенные элементы, что на первый взгляд может показаться лишним. Он содержит тэг <TITLE> (название), который в свою очередь включает #PCDATA. Я не стал упрощать эту конструкцию до тэга <!ELEMENT BYTITLE (#PCDATA), чтобы было ясно, что в тэге содержится параметр поиска, который обозначает тип поиска. В будущем мы сможем значительно расширить тип поиска и включить тэг <SUBTITLE> (подзаголовок). Это чисто концептуальное

решение, которое не следует общей практике XML. Любое описание DTD отражает взгляд автора на основную проблему, так что практически никогда не бывает единственно верного решения. Одним словом, не стесняйтесь экспериментировать!

Создание ответов

Каждый SQL-запрос отыскивает в базе данных MS Access имя автора, дату публикации статьи и имя XML-файла, содержащего статью. Чтобы отразить этот процесс, мы определили DTD, хотя формально и не будем использовать его в коде. Мы также не будем пользоваться функцией анализатора для проверки на состоятельность, так что DTD никогда не вступит в действие. Фактически код XML, который мы создаем, не устанавливает связь между XML и этим DTD. Каждая статья помечается тэгом `<paper>`:

```
<!ELEMENT PaperResponse (BYAUTHOR | BYTITLE | BYDATE | NoResults)*>
<!ELEMENT BYAUTHOR (paper)+>
<!ELEMENT BYTITLE (paper)+>
<!ELEMENT BYDATE (paper)+>
<!ELEMENT NoResults EMPTY>

<!ELEMENT paper (docURL?, title, author, pubdate)>
<!ELEMENT docURL (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT pubdate (#PCDATA)>
```

Данные, возвращенные сервером, могут содержать результаты более чем одного поиска. Например, если указан критерий для каждой группы на начальной странице, будет получен тэг `<PaperResponse>` (статья-ответ), в который заключены тэг `<BYAUTHOR>`, содержащий статьи, удовлетворяющие критерию по автору, тэг `<BYTITLE>`, содержащий статьи, удовлетворяющие критерию по названиям, и тэг `<BYDATE>` (по дате), включающий статьи, удовлетворяющие критерию по дате. Каждый поиск независим от других. Если никаких статей не найдено, будет возвращен тэг `<NoResults/>` (нет результатов).

XML-данные формируются функциями JavaScript. Функции возвращают клиенту XML-текст по мере его создания. Поскольку XML-данные передаются напрямую анализатору клиента, необходимо, чтобы HTML-тэги не передавались, иначе приложение клиента выйдет из строя, так как он не ожидает поступления HTML-тэгов.

Концептуальная схема задач, выполняемых в поисковой части нашего приложения клиент-сервер, представлена на рис. 10.4: функции JavaScript принимают критерии поиска, введенные пользователем, и компонуют правильную XML-строку. Строка добавляется к имени страницы `papersearch.asp` на стороне сервера и передается внедренному компоненту XML-анализатора. Анализатор запрашивает у сервера эту страницу, заставляя последний возвратить поток XML-данных, содержащих результаты проведенных поисков. Совершается обход дерева разбора, построенного анализатором, и создаются HTML-таблицы, которые атем динамически вставляются на страницу поиска.

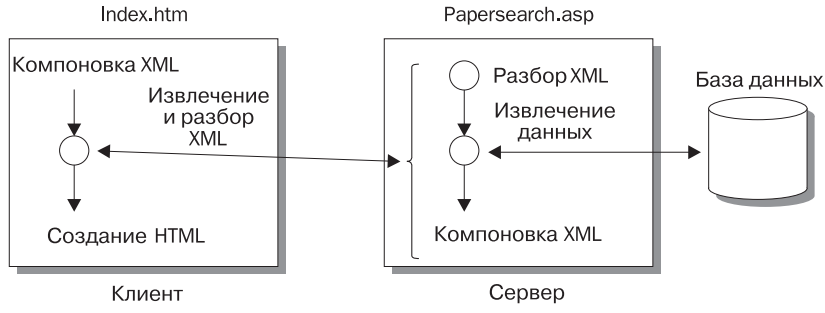


Рис. 10.4. Процесс посылки запроса, поиска и возвращения ответа

Совет *Наш простой пример не обеспечивает читателей механизмом для добавления новых статей на Web-сайт. Если вы решили потренироваться с кодом примера и хотите расширить его, а также пополнить базу данных, (см. http://webdev.wrox.co.uk/books/1525), вам придется обновить свою базу данных и добавить несколько файлов. Конкретно, вам понадобится графическое изображение автора (JPEG, GIF и т.п.), XML-файл, размеченный в соответствии с описанием DTD, и новая строка в таблице статей базы данных techpapers.mdb. Эта таблица содержит элементы для составляющих имени автора, названия, даты публикации и URL для указания местонахождения XML-версии статьи.*

Публикация статей

Хотелось бы, чтобы в момент вывода на экран статья была хорошо отформатирована. Этого можно добиться при помощи динамического HTML (Dynamic HTML, DHTML). Каждый материал, как уже было сказано, состоит из названия статьи, биографии автора и тела статьи. Тело статьи, в свою очередь, состоит из аннотации, за которой расположены абзацы и маркированные списки. Существует некоторый стандартный текст, например сведения об авторе, повторения которого в каждом XML-файле хотелось бы избежать. Можно ввести в него фотографию автора, но при этом не хотелось бы включать в XML-текст HTML-тэг `<IMG>`, чтобы не вставлять в XML информацию, относящуюся исключительно к визуальному представлению на компьютере клиента.

Для перевода XML-текста существуют правила XSL. Другое преимущество использования XSL состоит в том, что можно перемещать блок с информацией об авторе в пределах документа, не переписывая XML. Это можно сделать, изменяя правила в таблице XSL-правил. Чтобы осуществить подобную операцию, следует воспользоваться готовым XSL-процессором, выполненным в виде компонента ActiveX. Фактически, имея на сервере несколько функций JavaScript, мы можем использовать одну и ту же страницу, чтобы представить любой документ в нашей системе.

Схема на рис. 10.5 иллюстрирует прохождение потоков задач при поиске и отображении форматированной статьи с использованием XML. При щелчке по ссылке, находящейся в результатах поиска, запрашивается страница `ViewPaper.asp` с именем XML-файла, содержащего статью. Функции JavaScript, выполняемые на сервере, отыскивают имя файла и внедряют его в страницу, которая возвращается клиенту. Когда статья загружается на компьютере клиента, компонент XSL-процессора извлекает статью и таблицу XSL-стилей, содержащую правила форматирования. Эти правила применяются к статье, и в результате перед нами появляется форматированный HTML-документе. Теперь другая функция извлекает HTML-документ и вставлят его в страницу, просматриваемую пользователем.

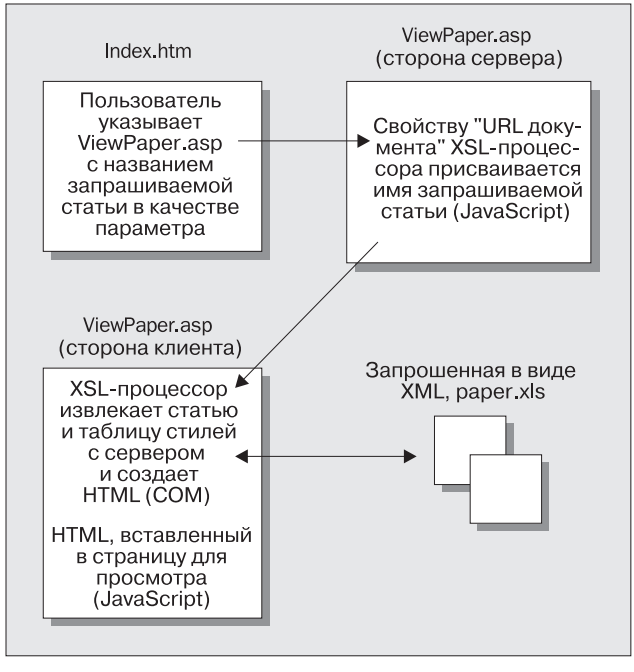


Рис. 10.5. Потоки задач при поиске и отображении форматированной статьи

Рассмотрение архитектуры ядра

Эта система безусловно будет работать, но насколько эффективна ее конфигурация? В отличии от плохо сконструированных автономных приложений, плохо сконструированная распределенная система затрагивает интересы всех пользователей сети. Применение XML-обработки на стороне клиента изменяет классическое распределение работы между клиентом и сервером, требуя, чтобы данные для представления форматировались у клиента. Передача от клиента серверу потоков XML-данных, потенциально имеющих значительный объем, поднимает вопросы, ответы на которые не очевидны. Перед тем как приступить к написанию кода приложения, следует рассмотреть проблемы, касающиеся архитектуры системы.

Компромиссы в системе клиент–сервер

Одна из проблем, часто упускаемых из виду при добавлении XML в систему клиент-сервер, – это дополнительная обработка, которая должна проводится на стороне клиента. Обычные приложения, использующие формы для получения от сервера полностью отформатированных HTML-данных, представляют собой весьма «тонкого» клиента. Возможности обработки и конфигурация на стороне клиента обычно весьма скромны. Использование XML в качестве механизма обмена данными требует проведения анализа на обоих концах линии связи и создания XML-документа на стороне клиента. Страница с параметрами поиска, которой пользуется клиент, содержит, как мы увидим, достаточно пространный код. Если предполагается использовать сервер только для работы с данным конкретным клиентом, этот дополнительный код не нужен. В конце концов, можно поступить следующим образом: сервер посылает HTML, тем самым избегая необходимости анализа XML и его преобразования в HTML на стороне клиента. Если отдать предпочтение этому варианту, то фактически можно полностью отказаться от XML, превратив нашу страницу в обычную HTML-форму. Используя XML для обмена данными, мы обеспечиваем независимость сервера от клиента. Это очень важно, принимая во внимание будущее поколение автоматизированных клиентов. Это полезно и для многих разработчиков Web-сайтов, поскольку позволяет легче управляться с различными возможностями браузеров. Вместо того, чтобы объединять форматирующие коды для разных платформ в один очень длинный код на сервере, можно поставить входную страницу, которая направляла бы клиентов на страницы поиска, соответствующие классу их браузера. Страницы с DHTML, созданным под специальные браузеры, могут работать с обычными клиентами. Очень «тонкие» клиенты с недостаточной памятью или слабыми возможностями обработки данных могут получать HTML, созданный XSL-процессором на сервере. Объем работы такой же, но мы изолировали корневой код, работающий с данными. Изменения вида представления данных могут проводиться независимо. Это исключает возможность возникновения побочных эффектов, влияющих на корневой код обработки данных при каких-либо действиях над кодом, меняющим вид представления информации.

В нашем примере мы решили организовать систему, доступную для будущих клиентов, чьи возможности и цели пока нельзя предугадать в полном объеме. Это оправдывает использование XML в качестве формата обмена данными и разметку самих технических статей с помощью XML. Сохранение разметки на всем пути от сервера до клиента позволит в будущем использовать программы поиска в промежуточном слое. Если бы нам понадобилось добавить аннотацию статьи к другим возвращаемым данным, мы могли бы использовать тот же сервер для создания списков литературы и библиотечных каталогов. Преимущества XML в обеспечении интеллектуального поиска документов будут сохраняться и для подобных списков.

Разве допуск клиента в основную базу данных и разрешение проводить там поиск лучше?

Безусловно, нет!

Сервер с базой данных иногда бывает сильно перегружен. Соединение пользователя с сетью может часто прерываться, или клиент может быть отделен от сервера линией с низкой скоростью передачи данных. Тогда, при невысоком темпе изменения информации, локальный поиск в достаточно регулярно обновляемых списках является привлекательной альтернативой поиску в самой базе данных. Подобная схема потребовала бы некоторого механизма, указывающего, как долго следует хранить список, прежде чем затребовать от сервера его обновленную версию. Агент, действующий от имени пользователя, безусловно, никогда не преобразует XML в HTML, но он мог бы выполнять некоторые другие операции. В любом случае, XML обеспечивает нам гибкость выбора. Поскольку наш сервер разработан для возврата XML-, а не HTML-документов, мы можем, не изменяя настройку, использовать его для этих новых приложений-агентов. Так как мы строим библиотечную систему надолго, разумно использовать в качестве механизма обмена данными именно XML. Он будет работать со многими типами клиентов, от читателя-человека до поисковой машины, проводящей автоматизированный поиск. Если бы нам пришлось разрабатывать приложение для однократного использования, мы бы придерживались базовых форм и HTML для поиска и ограничились бы использованием XML только для разметки документов.

Не существует единого ответа на вопросы, как распределить обработку и как долго сохранять значение данных. Цели проекта и компьютерное окружение диктуют способы, выбираемые при построении разветвленных систем. Даже новейшие способы, такие как XML, должны подчиняться базовым принципам разработки программного обеспечения.

Вопросы передачи данных

Как доставить XML от клиента к серверу для обработки? Протокол HTTP предназначен для передачи сколь угодно больших объемов данных, так что здесь у нас нет проблем. Клиенты, однако, традиционно передают много меньше. По существу, отправляются короткие указания, какой документ хочет видеть пользователь, а ведь XML – потенциально многословный язык. Означает ли это, что при разрешении данного противоречия возможны какие-либо трудности? Если они и существуют, есть три способа, от простого, но ограниченного до сложного, но мощного, – которые помогают решить задачу. Это:

- HTTP-запросы;
- асинхронная передача файлов;
- передача потоков вне HTTP.

HTTP-запросы

Самое простое решение – это сделать XML единственным параметром в HTTP-запросе. Приложения-формы посылают параметры в виде URL следующего вида:

```
somepage.asp?Param1=XXX$Param2=YYY
```

Взамен создается URL, в котором строка в XML-формате находится на месте параметров приложения-формы:

```
papersearch.asp?<PaperQuery><BYATHOR>...</PaperQuery>
```

При этом программирование на стороне сервера упрощается. Мы просто восстанавливаем параметры запроса и передаем их XML-анализатору.

Однако у данного способа есть два недостатка. Во-первых, хотя протокол HTTP 1.1 не ограничивает длину URL (для операции `GET`), серверы и представители пользователя имеют различные ограничения на максимальную длину. В этом случае ставится предел количеству запросов, которые можно передать за один раз. Хотя использование операции `POST` позволило бы избежать этих ограничений, но в нашем примере, к сожалению, поиск файла осуществляет внедренный компонент XML-анализатора, так что выбирать между `POST` и `GET` не в нашей власти. Здесь мы явным образом обмениваем функциональные возможности системы на простоту. Во вторых, наш анализатор использует стандартное HTTP-кодирование для URL. Иначе говоря, все небуквенно-цифровые символы преобразуются в escape-символ `%` и следующий за ним двусимвольный шестнадцатеричный код конвертируемого символа. Пробелы в названии статьи, которую мы ищем, будут заменены на `%20`, так что название `Deep Thoughts on Computing` (Размышления о вычислении) превратится в: `Deep%20Thoughts%20on%20Computing`. Расплачиваться за это придется серверу.

Передача файлов

Другой подход – это передача файла на сервер при помощи протокола FTP, а затем считывание его с диска в рамках процесса приема. Такое решение великолепно подходит при больших объемах передаваемых данных, но опять же приводит к появлению нескольких проблем. Во-первых, прямой синхронный обмен при этом варианте уже невозможен. Либо мы должны иметь некий фоновый процесс, занимающийся сканированием диска сервера и выявлением полученных файлов, либо клиент должен проинформировать сервер, что файл передан. Каждый из способов вызывает некоторые сложности при работе с нетривиальными системами. И, наконец, появляются проблемы с возвратом файла клиенту. В типичном случае клиент будет опрашивать сервер с периодическими интервалами, либо сервер уведомит клиента по e-mail. И все же сейчас существует много доступных программисту готовых разработок по FTP, так что отбрасывать этот способ не следует.

Программирование пользовательского сокета

Последний способ – установить прямое двунаправленное соединение, *сокет* (socket) вне протокола HTTP и осуществлять передачу между специализированным компонентом на стороне клиента и соответствующим компонентом на сервере. Казалось бы, это идеальное решение, но давайте не будем спешить и немного поразмыслим. Для начала нам пришлось бы разработать и реализовать свой собственный протокол, а подобная задача всегда считалась достаточно трудной. Хотя передачу данных и можно было бы осуществить через протокол FTP, но при этом возникают проблемы асинхронности, о которых мы уже говорили. Если мы прибегнем к программированию на заказ, нам придется потребовать более тесной связи между протоколом передачи и логикой нашего приложения. Далее, клиенты и серверы должны будут согласовать этот протокол, в сущности, сужая класс слабо связанных, широко распространенных систем, появление которых мы предвидим.

Дополнительные проблемы создают и брандмауэры. Большинство подобных механизмов сконфигурированы для опций обычной защиты, работающей с хорошо известными Internet-протоколами и портами. Наш протокол потребовал бы специальной конфигурации каждого брандмауэра, находящегося между клиентом и сервером. Если бы все клиенты, собирающиеся пользоваться нашим сервером, находились под нашим контролем, в этом еще был бы какой-то смысл. Мы могли бы использовать Java Applet на клиентской странице поиска для установления связи с сервером. Однако вряд ли нам в этом случае удалось бы реализовать сервер на языке скриптов. В роли сервера пришлось бы использовать компоненты Java, демоны Unix или сервисы Windows NT.

Наша простая система подразумевает небольшой объем данных, посылаемых на сервер. Давайте выберем простоту и займемся компоновкой и анализом XML.

Создание XML-файла на стороне клиента

Работа клиента начинается со страницы по умолчанию `index.htm`. Эта страница предоставляет собой пользовательский интерфейс, поддерживаемый функциями JavaScript, которые форматируют параметры поиска в XML-файл для передачи на сервер. Другие функции JavaScript обрабатывают ответ сервера.

Пользовательский интерфейс

Страница поиска состоит из нескольких достаточно типичных элементов HTML-формы, но не только из нее. Мы собираемся получать ввод пользователя прямо через объектную модель DHTML, реализованную в Internet Explorer и JavaScript. Это позволит нам контролировать, как, когда и в каком формате параметры поиска передаются на сервер. Поиск будем осуществлять по имени автора, названию статьи и дате публикации, но прежде всего сконцентрируемся на поиске по имени автора. В ячейки таблицы для форматирования помещаются три элемента ввода текста. Ниже приведен код HTML для части, касающейся поиска автора. Полный код этой страницы, `index.htm`, может быть загружен с нашего Web-сайта http://webdev.wrox.co.uk/books/1525.

```
<h3>By Author:</h3>

<table border="0" width="41%">
   <tr>
      <td width="41%" align="right"><font face="Verdana">First Name</font></td>
      <td width="59%"><input NAME="FNAME" ></td>
   </tr>
   <tr>
      <td width="41%" align="right"><font face="Verdana">Middle Initial</font></td>
      <td width="59%"><input SIZE="4" NAME="MI" > </td>
</tr>
   <tr>
      <td width="41%" align="right"><font face="Verdana">Last Name</font></td>
      <td width="59%"><input NAME="LNAME" ></td>
   </tr>
</table>
```

Имена элементов формы – FNAME, MI, LNAME – понадобятся нам позднее. Процесс поиска запускается, когда пользователь щелкает по клавише **Search** (Искать). Это тоже элемент формы, но с ним ассоциирован небольшой код:

```
<p align="left"><input TYPE="button" VALUE="Search" ONCLICK="OnSearch()"
NAME="SearchBtn">
</p>
```

Оформление параметров поиска

Как только пользователь щелкает по кнопке OnSearch(), программный код ONCLICK="OnSearch()" ассоциирует функцию JavaScript OnSearch() с кнопкой на странице и вызывает данную функцию. С этого момента начинается что-то интересное. Наступает момент компоновать XML для различных типов поисков, основываясь на содержании элементов формы.

```
function OnSearch()
{
   var sQueryXML = "";

// Проверяет, хочет ли пользователь искать "по имени" ('by name').
if (FNAME.value != "" || MI.value != "" || LNAME.value != "")
sQueryXML += makeByName(FNAME.value, MI.value, LNAME.value);

// Проверяет на поиск "по названию" ('by title').
if (TITLE.value != "")
sQueryXML += makeByTitle(TITLE.value);

// Проверяет на поиск "по дате публикации" ('publication date').
if (AFTER.value != "" || BEFORE.value != "")
sQueryXML += makeByPubDate(AFTER.value, BEFORE.value);

// Избегает прохода в оба конца по сети, если нет запроса поиска.
if (sQueryXML != "")
{
// Если есть хотя бы один запрос, обрамляет его корневыми тэгами.
sQueryXML = "<PaperQuery>" + sQueryXML + "</PaperQuery>";

// Присваивает документу XML-анализатора URL для запроса на сервер.
parser.async = "false";
parser.load("papersearch.asp?" + sQueryXML);

HandleResponse(document.all("results"));
   }
}
```

Для компоновки XML-документа, отсылаемого на сервер, мы используем строковую переменную sQueryXML. Необходимо также проверить каждый элемент формы на наличие параметров поиска. Если какие-нибудь параметры поиска найдены, они передаются функции JavaScript, которая возвращает их в виде XML. Если имеется хотя бы один запрос, его следует послать на сервер. Для этого нужен экземпляр COM XML-анализатора, чтобы обработать ответ, и мы размещаем один его экземпляр на странице со следующим HTML-кодом:

```
<object classid="clsid:E54941B2-7756-11D1-BC2A-00C04FB925F3"
ame="xmlDOM"></object>
```

Это просто невидимый компонент ActiveX. Класс `ID` ссылается на версию 5.0 ActiveX XML-анализатора. Эта версия тесно поддерживает объектную модель документа (W3C XML DOM, Level 1) в ее продвижении к статусу рекомендации версии 1.0. Строка

```
var parser = document.all("xmlDOM"); in OnSearch()
```

просто восстанавливает сноску на объект анализатора из текущей коллекции элементов страницы.

Если элемент ввода содержит какой-либо текст, мы компонуем XML для поиска соответствующего типа. Другими словами, если любой из текстовых элементов `FNAME`, `MI`, `LNAME` содержит хотя бы один символ, мы запрашиваем поиск по имени автора. XML для каждого типа поиска подсоединяется к переменной `sQueryXML`. Далее следует выделить объединительный код поиска по автору в отдельную функцию:

```
function makeByName(sFirst, sMid, sLast)
{
   // Открывает тэг "по автору".
   var sQueryString = "<BYAUTHOR>";

   if (sFirst != "")
       sQueryString += "<FNAME>" + sFirst + "</FNAME>";

   if (sMid != "")
       sQueryString += "<MI>" + sMid + "</MI>";

   if (sLast != "")
       sQueryString += "<LNAME>" + sLast + "</LNAME>";

   // Закрывает тэг "по автору".
   sQueryString += "</BYAUTHOR>";

   return sQueryString;
}
```

Этот фрагмент достаточно ясен. Значения соответствующих элементов ввода передаются в функцию. Для каждого элемента проверяется, содержит ли он какой-нибудь текст. Если да, то элемент обрамляется соответствующими разделителями тэгов и подсоединяется к строке, которая будет возвращена как значение функции. Все элементы содержатся внутри тэга `<BYAUTHOR>`, указывающего серверу тип поиска. Функции для других поисков, `MakeByTitle()` и `MakeByPubDate()`, аналогичны. При возвращении к `OnSearch()` мы завершаем XML корневым тэгом.

```
if (sQueryXML != " ")
{
sQueryXML = "<PaperQuery>" + sQueryXML + "</PaperQuery>";
...
```

XML, собранный для поиска по автору, как показано на рис. 10.3 нашего примера, выглядит следующим образом:

```
<PaperQuery>
```

```
    <BYAUTHOR>
        <FNAME>Stephen</FNAME>
        <MI>T</MI>
        <LNAME>Mohr</LNAME>
    </BYAUTHOR>
</PaperQuery>
```

Совет *Пуристы непременно заметят, что мы не используем анализатор для построения и проверки состоятельности XML по DTD. Мы также не проводим проверки состоятельности содержимого полей, как можно было ожидать в коммерческом коде, чтобы убедиться, что пользователь не ввел разметку. В приведенном выше примере задача ограничивалась только разработкой кода и для сервера, и для приложения. Следовательно, нам легко гарантировать, что в данном приложении сформирован именно состоятельный документ XML. В большинстве случаев код создается параллельно. Иначе говоря, здесь был выбран более легкий путь. Анализатор в Internet Explorer версии 5.0 при желании может быть установлен в режим верификации. Другое дело, когда коды на стороне клиентов и серверов пишутся разными, незнакомыми друг с другом программистами. Тогда действительно потребуется проверка XML на состоятельность. В таких случаях у нас не может быть уверенности, что все команды разработчиков действуют, исходя из одной версии DTD, так что появляется необходимость по мере обмена проводить формальную проверку состоятельности XML, чтобы избежать ошибок в понимании. Этот вопрос мы затронем, в одном из следующих разделов.*

Обработка результата, возвращенного сервером

Возвращаясь к функции `OnSearch()`, мы заканчиваем работу, получаем результаты поиска в виде XML и преобразуем их в HTML:

```
// Присваивает документу XML-анализатора URL для запроса на сервер.
        parser.async = "false";
        parser.load("papersearch.asp?" + sQueryXML);

        HandleResponse(document.all("results"));
```

Как был сделан переход от формирования запроса на поиск к обработке ответа? Метод анализатора `LOAD()` загружает файл, ассоциированный с переданным URL. Страница ASP `papersearch.asp` на сервере динамически создает XML в ответ на запрос, содержащийся в `sQueryXML`. Далее для запроса формируется URL, передается анализатору, и анализатор выполняет запрос. Когда возвращается метод `LOAD()`, в анализаторе либо находится дерево разбора, либо анализатор находится в состоянии ошибки. Прервем разработку на стороне клиента и последуем за нашим запросом на сервер.

Управление XML в Active Server Pages

Active Server Pages– это ответ Microsoft на CGI-скрипты. ASP представляют собой скриптовые страницы, исполняемые в процессах сервера, для повышения эффективности использования его ресурсов. Этот механизм обеспечивает лучшие характеристики, чем CGI. Исходный код Active Server Pages выглядит несколько иначе, чем для типичной Web-страницы. Обычно он представляет собой смесь свободно переплетенных кодов HTML и скриптов. В нашем случае `papersearch.asp` создает чистый XML-код, так что HTML-элементов нет совсем, отсутствуют даже тэги `<head>` и `<body>`. Перед тем как перейти к деталям, рассмотрим начало страницы `papersearch.asp`:

```
<%@ LANGUAGE = JavaScript %>
<%
var dbConn, dbRecordSet;
   /* Создает объект анализатора и извлекает запрос поиска XML из
   объекта Запрос (Request) сервера.*/
   var parser = Server.CreateObject("microsoft.xmldom");

parser.loadXML(Request.ServerVariables("QUERY_STRING"));
if (parser.readyState == 4 && parser.parseError.reason == "")
{
var docroot = parser.documentElement;
       if (docroot.nodeName == "PaperQuery")
       {
           // Устанавливает глобальную связь.
dbConn = Server.CreateObject("ADODB.Connection");
           dbConn.Open("TechPapers", "server", "");
           dbRecordSet = Server.CreateObject("ADODB.RecordSet");
           // Обрабатывает запрошенные поиски.
AssembleQueries(docroot);
           // Очищает ресурсы.
           dbRecordSet.Close();
           dbConn.Close();
           dbConn = null;
           dbRecordSet = null;
       }
       else
           Response.Write("<InvalidQuery></InvalidQuery>");
       }
       else
           Response.Write("<Error></Error>");
```

Тело функции JavaScript аналогично функции `main()` в обычном приложении на языке C. Оно выполняется, когда ASP входит в страницу. Директива `<%@ LANGUAGE=JavaScript %>` устанавливает для страницы язык скриптов сервера по умолчанию. Но сейчас вместо статического размещения на странице компо-

нента анализатора мы создадим его в скрипте. `Server` – это глобальный объект, предлагаемый реализацией ASP для Microsoft Internet Information Server (IIS) с целью обеспечения коду ASP доступа к функциям сервера. Метод `CreateObject()` приказывает объекту сервера создать экземпляр нового компонента ActiveX.

Совет *В этом случае также подошла бы строка "new ActiveXObject ("microsoft. xmldom")". Вовлечение сервера в процесс создания является хорошей привычкой, которую полезно было бы приобрести. Если мы хотим попробовать применить некоторую оптимизацию, такую, например, как сохранение объекта сверх времени сеанса пользователя или ASP приложения в IIS 4.0, мы должны позволить серверу создать экземпляр объекта, чтобы он мог должным образом управлять его временем жизни.*

Глобальные объекты сервера

Два других полезных объекта, предоставляемых ASP, это `Request` и `Response`. Объект `Request` позволяет получить информацию о запросе, переданном на данную страницу. `Request.ServerVariables` представляет собой совокупность переменных, переданных при запросе. В нашем случае:

```
Request.ServerVariables("QUERY_STRING")
```

получает полную XML-строку, сформированную на стороне клиента для кодирования параметров поиска. `Response` – это объект, представляющий поток, возвращаемый клиенту. XML-ответ со стороны сервера может быть значительно длиннее, чем XML-запрос со стороны клиента, поскольку пользователи в принципе могут работать с информацией для многих названий. Не следует держать все это в одной строке. Следовательно, мы будем вызывать `Response.Write()` всякий раз, когда у нас появится какой-либо фрагмент XML и позволим реализации ASP позаботиться о буфере.

Загрузка XML-строки

Наш способ загрузки анализатора XML-данными несколько отличается от использованного на стороне клиента. На этом этапе необходимо, чтобы клиент отыскал динамически созданный файл, указанный URL. Сейчас у нас есть XML-строка, (строка запроса) и мы хотим загрузить ее в анализатор. Для этого можно использовать метод анализатора `loadXML()`. Анализатор принимает XML-строку как законченный файл и пытается проанализировать ее.

Совет *Предыдущий анализатор Microsoft COM обладал аналогичной способностью, реализованной через COM-интерфейс IPersistStreamInit, но в этом случае строку нельзя было вызвать через интерфейс, доступный из скриптов. Приходилось писать вспомогательный компонент ActiveX, который принимал XML-строку и загружал ее в анализатор. Наличие метода loadXML()является основным усовершенствованием для программирования на стороне сервера.*

Получение корня дерева разбора

По окончании работы метода `loadXML()` можно предположить, что у нас имеется дерево разбора или состояние ошибки. Когда только анализатор закончит загрузку XML, свойству `readyState` будет присвоено значение 4. Но это еще не дает полной уверенности, что XML был успешно проанализирован, а всего лишь означает, что во время загрузки не произошло никаких фатальных ошибок. Поэтому также следует обратиться к объекту `lastError` и убедиться, что свойство `reason` пусто.

```
if (parser. readyState == 4 && parser.lastError.reason == " ")
```

При выполнении обоих этих условий работа на сервере продолжается, и можно с уверенностью сказать, что, по крайней мере, получен корневой узел. Эта операция проводится с помощью метода анализатора `documentNode()`. Здесь необходимо отметить, что объекты-узлы обладают тремя важными свойствами, имеющими существенное значение для нашего проекта, а именно:

- `nodeType` – узлы `ELEMENT` имеют тип `0`;
- `nodeName` – строка с именем тэга, то есть `PaperQuery` для `<PaperQuery>`;
- `nodeValue` – текст, содержащийся внутри тэга (включая внедренные тэги).

Теперь необходимо провести быструю проверку имени корневого узла, чтобы убедиться, что получен XML-документ, предназначенный для данного сервера. В нашей системе корнем является `<PaperQuery>`. Если проверка не удалась, клиенту будет отослан некий правильный XML-код, указывающий на ошибку, и он проанализирует у себя полученный XML-файл (мы вернемся к этому вопросу позже). Если же клиент сформировал корректный запрос на поиск, можно идти дальше по дереву разбора и обращаться к базе данных.

Подготовка ресурсов базы данных

При этом возникает необходимость установить глобальную связь с нашей базой данных, что позволит избежать повторения кода в различных функциях, специфичных для типов поиска. Кроме того, связь подобного типа способна устранить непроизводительные издержки на многократное установление и прерывание соединения.

```
dbConn = Server.CreateObject("ADODB.Connection");
dbConn.Open("TechPapers", "server", "");
dbRecordSet = Server.CreateObject("ADODB.RecordSet");
```

Совет *В нашем примере используется база данных Microsoft Access. Читатели, имеющие опыт работы в Access, должны отметить, что ADO требует символ шаблона % (wildcard character), соответствующий SQL-92, а не звездочку (*), как это принято в Access.*

Обход дерева разбора

Функция `AssembleQueries()` является обратной функции `OnSearch()`, существующей на странице клиента. Она обходит непосредственных потомков корневого

узла дерева разбора и управляет конкретной обработкой, основанной на типе запрошенного поиска.

```
function AssembleQueries(oRoot)
{
   var childNode;
   var sQueryStem = "SELECT Papers.Title, Papers.PaperURL, Papers.PubDate,
Papers.AuthorFirst, Papers.AuthorMI, Papers.AuthorLast FROM Papers ";

   Response.Write("<PaperResponse>");  // Создает корень XML-документа.

   // Каждая дочерняя запись представляет отдельный поиск в базе данных.

   for (var nChild = 0; nChild < oRoot.childNodes.length; nChild++)
   {
   childNode = oRoot.childNodes.item(nChild);
       if (childNode.nodeType == 1)
       {
           if (childNode.nodeName == "BYAUTHOR")
               AssembleAuthor(sQueryStem, childNode);

           if (childNode.nodeName == "BYTITLE")
               AssembleTitle(sQueryStem, childNode);

           if (childNode.nodeName == "BYDATE")
               AssemblePubDate(sQueryStem, childNode);
       }
   }
   Response.Write("</PaperResponse>");
}
```

Каждый узел содержит коллекцию `childNodes`, включающую сноски на все дочерние узлы. Свойство `childNodes.length` показывает число дочерних узлов, при этом каждый индивидуальный дочерний узел отыскивается путем передачи его порядкового индекса в метод `childNodes.index()`. Любой поиск, затребованный клиентом, представляется в виде XML-элемента, расположенного вблизи корня. Если пользователь указал параметры для всех трех типов поиска, то корень будет иметь в качестве дочерних узлов элементы `<BYAUTHOR>`, `<BYTITLE>`, `<BYDATE>`.

Поскольку в нашем случае способность анализатора проверять XML на состоятельность не используется, нам придется смириться с тем фактом, что приходящие XML-данные могут оказаться несоответствующими этому критерию. Хотелось бы также, чтобы вместе с поиском было предоставлено условие совместимости снизу вверх. Другими словами, клиент, разработанный с применением более позднего описания DTD, чем определение, используемое на сервере, передаст и некие относительно старые элементы, которые сервер способен понять и обработать. Такое, очевидно, вполне может случиться в слабо связанных системах в Internet. Например, некая группа разработала DTD для задач определенного типа, а клиенты и серверы были сконструированы другими организациями, так что возможна ситуация, в которой более современный клиент может потребовать услуг от устаревшего сервера. Можно было бы настоять на проверке состоятельности и отвергнуть все, что не удовлетворяет определению типа документа (DTD)

сервера, однако при подобном подходе клиенты, нужды которых можно было бы до некоторой степени удовлетворить, будут лишены обслуживания. В таком случае серверу следует понизить свой уровень. С этой целью просто пропустим все дочерние узлы, которые не имеют типа ELEMENT или имеют тип ELEMENT но с незнакомыми именами. Например:

```
if (childNode.nodeType == 0)
       {
           if (childNode.nodeName == "BYAUTHOR")
```

До тех пор пока клиент помещает все свои запросы на поиск непосредственно при корне документа, сервер будет отвечать на три типа поиска, которые он распознает.

Совет *Устойчивость системы обслуживания можно также повысить, если во время формирования поисков отказаться заходить в любые незнакомые узлы. Это позволило бы будущим версиям DTD вставлять их внутрь тэга с содержанием. Таким образом, сервер обнаруживал бы все поиски, которые способен распознать, где бы они ни встретились внутри XML-запроса, и пропускал только неизвестные ему имена. Для упрощения мы откажемся от этого дополнительного критерия совместимости снизу вверх.*

Припомните, наш поиск по-прежнему ведется по имени автора. Предположим, что сервер получил тэг <BYAUTHOR>. Нам следует проанализировать этот тэг, выделить параметры поиска и сформировать соответствующий SQL-запрос.

Извлечение параметров

Функция JavaScript AssembleAuthor() в странице papersearch.asp вызывается со строкой, содержащей оператор SELECT нашего SQL-запроса, и узлом элемента <BYAUTHOR>.

```
function AssembleAuthor(sStem, oChild)
{
   var paramNode;
   var sFirst = "", sMI = "", sLast = "";
   var sConstraintClause = "", sQuery = "";
   var nParam = 0;

   for (nParam = 0; nParam < oChild.childNodes.length; nParam++)
   {
       // Извлекает параметры в понятной форме.
       paramNode = oChild.childNodes.item(nParam);

       if (paramNode.nodeType == 0)
       {

       if (paramNode.nodeName == "FNAME")
           sFirst = "AuthorFirst LIKE '"+ paramNode.nodeValue + "'";

       if (paramNode.nodeName == "MI")
           sMI = "AuthorMI LIKE'"+ paramNode.nodeValue + "'";
```

```
    if (paramNode.nodeName == "LNAME")
            sLast = "AuthorLast LIKE '" + paramNode.nodeValue + "'";
    }
}

    // Строит из переданных параметров предложение SQL WHERE.
    sConstraintClause = sFirst;

    if (sConstraintClause != "" && sMI != "")
        sConstraintClause += "AND";

    sConstraintClause += sMI;

    if (sConstraintClause != "" && sLast != "")
        sConstraintClause += "AND";

    sConstraintClause += sLast;

    // Заканчивает ограничительное предложение и выполняет запрос.
    if (sConstraintClause != "")
        sConstraintClause += "WHERE" + sConstraintClause;
    sQuery = sStem + sConstraintClause + ';';

    MakeResponse(1, sQuery);
}
```

Обратите особое внимание на этот код. Элемент <BYAUTHOR> определен в DTD сервера в виде:

```
<!ELEMENT BYAUTHOR (FNAME?,MI?,LNAME?)>
```

Теперь проверим все подчиненные узлы выписанного выше узла и убедимся, что каждый из них является типом ELEMENT, проконтролируем также их имена; все они должны представлять собой распознаваемый тэг.

```
paramNode = oChild.childNodes.item(nParam);

        if (paramNode.nodeType == 1)
        {
            if (paramNode.nodeName == "FNAME")
```

Наши усилия и на этот раз должны быть сконцентрированы на том, чтобы связь между клиентом и сервером осталась настолько свободной, насколько возможно. Опять же состоятельность XML нас в этот момент не волнует. Подобный подход вполне согласуется с философией Internet и нашим представлением о том, каким образом агенты с несовершенным знанием друг о друге должны взаимодействовать между собой. Будущие, более широкие определения DTD для данного приложения могли бы разрешить поиск по возрасту автора или по данным о его членстве в различных обществах. Наш сервер пока не способен распознать подобные параметры, но он вполне готов ответить на параметры, касающиеся имени.

Поскольку возможна любая комбинация элементов имени, каждое условие оператора WHERE будем компоновать индивидуально. После того, как появятся все параметры, содержавшиеся в запросе <BYAUTHOR>, следует объединить их и закончить предложение. Объединение произойдет после того, как будет закончен цикл по коллекции дочерних узлов. Теперь нам необходимо убедиться, что SQL-запрос

скомпонован верно. Сформировав запрос в переменную `sQuery`, можно осуществить запрос и в базе данных. Такой код является общим для всех типов поисков, так что мы произвольно задаем нумерацию типа поиска для первого параметра функции `MakeResponse()` в следующем виде:

- `1-BYAUTHOR`;
- `2-BYTITLE`;
- `3-BYDATE`.

Если нумерация покажется вам несколько необычной, вспомните, что мы сконцентрировали внимание на поиске по автору. Полный код, доступный для загрузки с http://webdev.wrox.co.uk/books/1525, содержит также поиски по дате публикации и по названию. Второй параметр в вызове функции – это строка, содержащая SQL-запрос.

Получение ответов на запросы пользователя

Теперь, когда у нас есть SQL-запрос, формально готовый задать вопросы, на которые клиент хочет получить ответы, можно приступать к выполнению запроса – точнее, провести итеративную обработку курсора базы данных и создать XML-документ для возврата клиенту. Действия над базой данных и XML мы связали воедино, чтобы избежать хранения между получением ответов слишком большого объема данных в виде строк. Идея состоит в том, чтобы взять параметры для единичного поиска, поработать с базой данных и как можно скорее отослать клиенту полученные XML-данные. Посылать ли XML-данные клиенту немедленно, зависит от конфигурации буфера на сервере HTTP. Мы свою работу выполнили и теперь можем забыть об этой части ответа.

Получение данных при помощи ActiveX Data Objects

Предположим, что для работы с базой данных было решено использовать ActiveX Data Objects (ADO). Это относительно простая и плоская объектная структура, которая в будущем скорее всего станет стандартом Microsoft для выполнения подобного типа работ в Windows. На данном моменте останавливаться особо не будем, потому что вы могли бы использовать любой API, с каким предпочитаете работать.

Вот как выглядят первые несколько строк функции `MakeResponse()`:

```
function MakeResponse(nQueryType, sQuery)
{
    var sTypeEndTag;

    dbRecordSet = dbConn.Execute(sQuery);

    if (dbRecordSet.EOF && dbRecordSet.BOF)
        Response.Write("<NoResults></NoResults>");
```

Это достаточно простая операция. Для ее выполнения необходимо обратиться к глобальному соединению с базой данных с командой выполнить наш запрос и в ответ принять группу выбранных записей (recordset). Она эквивалентна курсору базы данных в SQL. Если группа записей пуста, оба свойства `BOF` и `EOF` (соответственно

начало и конец файла) истинны. Если была возвращена по крайней мере одна запись, курсор будет установлен на одну из этих возвращенных записей, а это означает, что ни свойство BOF, ни свойство EOF не будут истинными. Если группа записей пуста, записываем тэг <NoResults>, чтобы проинформировать клиента, что по его параметрам поиска соответствующих записей в базе данных не найдено.

Теперь рассмотрим второй вариант ответа. Итак, на данный момент нам известно, что в группе выбранных записей имеется несколько строк. После записи тэга типа поиска обратно в поток для передачи клиенту следует провести цикл по группе выбранных записей, записывая тэг <paper> для каждой записи. Если не касаться пока строк, записывающих XML, это будет выглядеть примерно так:

```
while (!dbRecordSet.EOF)
    {
    // Здесь расположен некоторый код, создающий XML.

    dbRecordset.MoveNext( );
    }
```

Динамическая генерация XML

Интересующей нас частью этого приложения является XML, так что его мы и рассмотрим. Ниже полностью приведена функция MakeResponse():

```
function MakeResponse (nQueryType, sQuery)
{
    var sTypeEndTag;

    dbRecordSet = dbConn.Execute(sQuery);

    if (!dbRecordSet.BOF)
        dbRecordSet.MoveFirst( );

    if (!dbRecordSet.EOF)
        Response.Write("<NoResults></NoResults>");

// nQueryType является произвольным и полностью внутренним для данной страницы.
switch (nQueryType)
{
    case 1:
    Response.Write("<BYAUTHOR>");
    sTypeEndTag = "</BYAUTHOR>";
    break;

    case 2:
    Response.Write("<BYTITLE>");
    sTypeEndTag = "</BYTITLE>";
    break;

    case 3:
    Response.Write("<BYPUBDATE>");
    sTypeEndTag = "</BYPUBDATE>";
    break;
    }

    /* Совершается шаг по набору записей с построением поддерева <Paper>..
```

```
        ..для каждой возвращенной строки. */
    while (!dbrecordSet.EOF)
    {
        Response.Write("<Paper>");
        if (dbRecordSet("PaperURL") != "")
                Response.Write("<docURL>" + dbRecordSet("PaperURL") + "</docURL>");
            if (dbRecordSet("Title") != "")
                Response.Write("<title>" + dbRecordSet("Title") + "</title>");
            // Собираем из частей полное имя.
            sName = dbRecordSet("AuthorFirst") + "" +
                dbRecordSet("AuthorMI") + "" + dbRecordSet("AuthorLast");
            if (sName != "" && sName != "")
                Response.Write("<author>" + sName + "</author>");
            if (dbRecordSet("PubDate") != "")
                Response.Write("<pubdate>" + dbRecordSet("Pubdate") + "</pubdate>");
            Response.Write("</paper>");
            dbRecordSet.MoveNext( );
        }
        Response.Write(sTypeEndTag);
    }
```

Сначала мы записываем тэг типа поиска (в нашем примере `<BYAUTHOR>`) и запоминаем соответствующий закрывающий тэг, чтобы использовать его позже. Затем открываем тэг `<paper>` и пишем тэги для любого поля в непустой записи.

```
Response.Write("<Paper>");
if (dbRecordSet("PaperURL") != "")
    Response.Write("<docURL>" + dbRecordSet("PaperURL") + "</docURL>");
if (dbRecordSet("Title") != "")
    Response.Write("<title>" + dbRecordSet("Title") + "</title>");
...
```

Поскольку наши данные являются самописывающимися, вставлять заполнители взамен пропущенных данных не обязательно. Закончим запись закрывающим тэгом `</paper>`. Когда будут сформированы тэги `<paper>` для всех записей, входящих в нашу группу выбранных записей, запишем закрывающий тэг для данного типа поиска. Возвращаясь по стеку вызовов, вспомним, что функция `AssembleQueries()` выполняет следующую строку:

```
Response.Write("</PaperResponse>");
```

чтобы по возвращении из этой функции XML-код ответа был закрыт.

Мы закончили создание XML-файла, который нужен для удовлетворения запроса клиента. Текст XML, созданный для того поиска (по имени автора), который показан на рис. 10.1 в начале этой главы, выглядит так:

```
<PaperResponse>
    <BYAUTHOR>
        <paper>
```

```
            <docURL>manifesto.xml</docURL>
            <title>A Future for Networked devices</title>
            <author>Stephen T Mohr</author>
            <pubdate>Tue Jun 30 00:00:00 EST 1998</pubdate>
        </paper>
    </BYAUTHOR>
</PaperResponse>
```

Как реагирует клиент

Вспомним, что процесс поиска и формирования ответа на стороне сервера начался после установки свойства URL объекта-анализатора на стороне клиента. Анализатор запросил файл, загрузил возвращенный с сервера файл, проанализировал его и создал дерево разбора (предполагается, что файл был правильным). Две последние строки функции `OnSearch()` на стороне клиента выглядели так:

```
parser.load("papersearch.asp?" + sQueryXML);
HandleResponse(document.all("results"));
```

Именно эти строчки побудили компонент-анализатор сделать запрос на сервер. Результатом поиска оказался документ XML-данных, созданный Active Server Page. Компонент загрузил возвращенный документ и проанализировал его. В функции `HandleResponse()` (обработка ответа) мы конвертируем XML в HTML для представления пользователю.

Подготовка XML-файла к разбору на стороне клиента

Вся совокупность объектов, соответствующих HTML-элементам на вашей странице, хранится в Internet Explorer. Передавая имя области `<DIV>`, в которой мы хотим разместить результаты, получаем ссылку на соответствующий объект:

```
document.all("results");
```

Эта сноска переносится функцией `HandleResponse()`. Код для проверки статуса анализатора уже должен быть вам знаком.

```
function HandleResponse(oDIV)
{
    var docroot;
    var nChildCount;
    var sTable = "";

    // Очищает область результатов на странице.
    oDIV.innerHTML = "";

    /* Если анализатор готов, проверяет корень на правильные типы. */

if (parser.readyState == 4 && parser.parseError.reason == "")
    {
        docroot = parser.documentElement;

        if (docroot.nodeName == "PaperResponse")
        {
            for (nChild = 0; nChild < docroot.childNodes.length; nChild++)
```

```
            {
                sTable = HandleResults(docroot.childNodes.item(nChild));

                /* Эта строка будет пустой, если дочерний узел окажется не того
типа, с которым мы работаем. Если имеется таблица, вставляем ее на страницу.
*/

                if (sTable != "")
                    oDIV.insertAdjacentHTML("beforeEnd", sTable);
            }
        }
        else
/* Неверный тип ответа; согласуйте с администратором сервера. */
            window.alert("Improper response type; check with server
administrator.");
    }
    else
/* Анализатор не готов или ответный документ неправильный. */
/* Убедитесь, что вы обращаетесь к нужному серверу. */
        window.alert("Parser not ready or response not well-formed." +
            "Ensure you are contacting an appropriate server.");
```

После очистки `results <DIV>`, проверяем, не находится ли анализатор в состоянии ошибки. Если все в порядке, получаем корневой узел:

```
oDIV.innerHTML = "";

if (paper.readyState == 4 && parser.lastError.reason == "")
{
    docroot = parser.documentNode;
...
```

Если в корне находится тэг `<PaperResponse>`, который мы и ожидали увидеть, продолжаем обработку. Передаем каждый узел из числа прямых потомков в функцию `HandleResults()`. Если эта функция создает HTML-таблицу, помещаем ее в область результатов страницы.

```
if (docroot.nodeName == "PaperResponse")
{
    for (nChild = ); nChild < docroot.childNodes.length; nChild++)
    {
        sTable = handleResults(docroot.childNodes.item(nChild));

        if (sTable ! = "")
            oDIV.insertAdjacentHTML("beforeEnd", sTable);
...
```

В нашем примере потомком является тэг `<BYAUTHOR>`. Вспомним, что в соответствии с заранее определенным DTD непосредственными потомками корневого узла являются тэги типов поиска `BYAUTHOR`, `BYDATE`, `BYTITLE`. До сих пор код был очень похож на тот, который мы наблюдали на сервере – на это есть особая причина. Обе выполненные задачи анализа представляют собой зеркальные подобия, и COM-компонент анализатора ведет себя одинаково на обеих сторонах связки клиент-

сервер. Если будет выбрана другая методика проведения анализа, вы можете встретиться с некоторыми проблемами, которые зависят от выбранного типа сервера и технологии. В этом смысле комбинация JavaScript и ActiveX очень удобна для программистов.

Правда, в обработке ошибок существует небольшое отличие. На сервере не было непосредственного пользователя, и нам приходилось передавать ошибки в видеXML-тэгов. Поскольку сейчас функция выполняется в браузере, мы можем воспользоваться DHTML для предоставления пользователю диалогового окна (рис. 10.6).

Рис. 10.6. Диалоговое окно анализатора в клиентском приложении

Подготовка к работе с результатами

Казалось бы, все уже готово для воспроизведения HTML-документа, однако здесь есть небольшая «тонкость». Страница клиента позволяет пользователю за один раз передать три различных запроса на поиск. Неплохо было бы проверить тип поиска в XML-ответе и предоставить пользователю визуальный признак, указывающий, какие результаты какому запросу соответствуют. Именно это и осуществляется в функции `HandleResults()`:

```
function HandleResults(node)
{
   var i;
   var sReply = "";
   var child;

   if (node.nodeType == 1) // Это XML ELEMENT
   {
       if (node.nodeName == "BYAUTHOR")
           sReply = '<h2>Results from Author</h2>';

if (node.nodeName == "BYTITLE")
sReply = '<h2>Results from Title</h2>';

if (node.nodeName == "BYPUBDATE")
sReply = '<h2>Results from Publication Date</h2>';

if (sReply != "")  // Имеется набор результатов, которые мы распознаем.
{
sReply += '<table border="1" width="100%">';

for (i = 0; i < node.childNodes.length; i++)
{
```

```
/* Цикл по узлам-потомкам, реагируем только на то, что можем распознать, то есть
на статьи (papers). */
                child = node.childNodes.item(i);
                if (child.nodeType == 1)
                {
                    if (child.nodeName == "Paper")
                        sReply += MakeHTMLFromPaper(child);

                    if (child.nodeName == "NoResults")
                        sReply += '<tr><td width="100%">No papers found</td></tr>';
                }
            }

            // Закрываем таблицу.
            sReply += '</table>';

}
    }
    return sReply; // Может быть пустым.
}
```

Мы проверяем тип подчиненного узла и если распознаем этот тип, то создаем заголовок второго уровня. Таким образом, для поиска по имени автора имеем:

```
if (node.nodeType == 0)
    {
    if (node.nodeName == "BYAUTHOR")
        sReply = '<h2>Results from Author</h2>';
...
```

Теперь можно перейти к следующему уровню дерева разбора. Именно здесь следует ожидать встречу с тэгами `<paper>`. Если дождались, передаем управление функции `MakeHTMLFromPaper()`. При этом, однако, следует помнить, что сервер посылает пустой элемент `<NoResults></NoResults>`, если никаких записей из базы данных возвращено не было. В этом случае пишем HTML-код для таблицы с одной строкой (`<tr>`) и одной ячейкой (`<td>`), содержащей текст `No papers found` (Статьи не найдены).

```
if (child.nodeName == "Paper")
    sReply = sReply + MakeHTMLFromPaper(child);
if (child.nodeName == "NoResults")
    sReply = sReply + <tr><td width="100%">No papers found</td></tr>';
...
```

Заполнение таблицы

Наконец можно считать, что все возможные трудности остались позади. К настоящему моменту мы добрались до функции `MakeHTMLFromPaper()`, нам известно, что у нас есть результаты и мы знаем, какой тип поиска дал эти результаты. Теперь можно предоставить эту информацию пользователю. Первая часть функции очень напоминает выражение, которое уже было представлено на сервере. Каждый узел изучается, определяется, что он собой представляет, после чего его сохраняют

в соответствующей строке, чтобы позднее собрать их все в одну большую строку. На этом этапе была задействована самоописываемость XML-данных, предоставляющих некоторую гибкость. Если тэги приходят с нужного уровня дерева разбора, но в несколько другом порядке, у нас все равно есть возможность вставить нужную информацию в соответствующую ячейку таблицы. Можно было бы настоять на использовании анализатора, осуществляющего сверку состоятельности с DTD, но лучше воспользоваться несколько неформальным подходом с тем, чтобы приспособиться к незначительным, не фатальным ошибкам реализации.

```
function MakeHTMLFromPaper(paperNode)
{
   var i;
   var childNode;
   var sRow = "";
   var sPubTitle = "", sPubLink = "", sAuthor, sPubDate;

   for (i = 0; i < paperNode.childNodes.length; i++)
   {
/* Проходим циклом по всем дочерним объектам и получаем необходимые части данных,
в каком бы порядке они ни появлялись. */

childNode = paperNode.childNodes.item(i);

   if (childNode.nodeType == 0) // Проверяем на узел ELEMENT.
   {
       if (childNode.nodeName == "title")
           sPubTitle = childNode.nodeValue;

       if (childNode.nodeName == "docURL")
           sPubLink = childNode.nodeValue;

       if (childNode.nodeName == "author")
           sPubAuthor = childNode.nodeValue;

       if (childNode.nodeName == "pubdate")
           sPubDate = childNode.nodeValue;
   }
}
```

Получив все необходимые части, запишем их в виде HTML. Чтобы ячейки таблицы были красиво расположены, можно включить атрибут `width` в тэг ячейки (`<td>`). Желательно, чтобы каждая ячейка занимала приблизительно треть ширины таблицы.

```
/* У нас есть все части данных. Теперь сформатируем строку в таблице HTML. */

sRow += '<tr><td width="34%">';

/* Первая ячейка является ссылкой, если был передан URL, в противном случае просто
текст. */

if (sPubLink != "")
{
   sRow += '<a href="Viewpaper.asp?' + sPubLink + '">';
   if (sPubTitle !="")
```

```
        sRow += sPubTitle + '</a>';
    else
        sRow += 'Untitled paper' + '</a>';
}
else
{
if (sPubTitle != "")
    sRow += sPubTitle;
}

sRow += '</td><td width="33%">';
if (sAuthor != "")
    sRow += sAuthor;
sRow += '</td>';

sRow += sRow     = '<td width="33%">';
if (sPubDate != "")
    sRow += sPubDate;
sRow += '</td></tr>';

return sRow;
}
```

Код, написанный для первой ячейки, выглядит несколько усложненным. Если в базе данных статья находится вместе с URL-указателем на XML-файл (в котором находится сама статья), необходима ссылка. Вот как выглядит тэг привязки (якорь, анкер) в HTML: `<a href=ViewPaper.asp?XML_paper>`. Активная страница `ViewPaper.asp` отличается от той ASP, какую мы использовали на сервере. Она поможет нам справиться с преобразованием из XML и XSL в HTML. Если название статьи окажется без URL, другими словами, если копии статьи у нас нет, выведем название в виде простого текста.

```
if (sPubLink != "")
{
    sRow += '<a href="ViewPaper.asp?' + sPubLink + '">';
    if (sPubTitle != "")
        sRow += sPubTitle + '</a>';
    else
        sRow += 'Untitled paper' + '</a>';
}
else
{
    if (sPubTitle != "")
        sRow += sPubTitle;
}
...
```

Две оставшиеся ячейки для результатов поиска по имени автора и дате публикации не требуют особых пояснений.

Скомпоновав HTML-строку для одной строки таблицы, возвращаем ее в вызывающую функцию `HandleResults()`. Она подсоединяет результат к хранимой ею строке, содержащей HTML-код для всего поиска. Эта строка возвращается фун-

кции HandleResponse(), которая добавляет ее в конец содержимого, находящегося в области <DIV>:

```
sTable = HandleResults(docroot.childNodes.item(nChild));

if (sTable != "")
   oDIV.insertAdjacentHTML("beforeEnd", sTable);
```

Таким образом, шаг за шагом, поиск за поиском, результаты предоставляются пользователю.

Представление материалов

Рано или поздно пользователь решит посмотреть саму статью, однако вывод необработанных XML-данных непременно вызовет у него чувство разочарования, не говоря уже о том, что их просто будет трудно читать. Мы уже создали в нашем коде большой DHTML, и сейчас для разнообразия желаем, чтобы за нас поработал кто-то другой. К счастью, у нас хватило сообразительности, чтобы выбрать в качестве способа представления данных расширяемый язык таблиц стилей (XSL).

Итак, существует несколько способов связывания стилей презентации с XML-данными. Две методики считаются основными, к ним относятся каскадные таблицы стилей (CSS) и расширяемый язык таблиц стилей (XSL). Каскадные таблицы стилей пока более разработаны, хотя реализация их на практике совместно с XML-файлами в основных Web-браузерах только начинается. Язык XSL, в свою очередь, тоже находится в консорциуме W3C и тоже на стадии разработки. Поскольку в этой книге мы ставим задачу испытать самые новые программные механизмы, мы и на этот раз выберем способ представления, исходя исключительно из его возможностей, а не из опыта реального применения.

Каскадные таблицы стилей легче конструировать, но они не позволяют выполнить что-то большее, чем сопоставление стилистических элементов с тэгами XML. Язык XSL, напротив, является системой, основанной на содержании тэгов XML и управляемой правилами. XSL создает исходный код источника, основанный на тэгах XML в данном контексте. Поскольку оба способа описаны в главах 7 и 8 этой книги, здесь ограничимся обсуждением конкретного использованием XSL, для чего приведем короткий пример. Дальнейшую информацию по таблицам стилей можно найти в книге Professional Style Sheets for HTML and XML, Wrox Press (ISBN 1-1861001-65-7).

XSL позволяет генерировать текст, который даже не упоминался в оригинальном XML. Например, у нас есть желание предоставить стандартные заголовки для аннотации и информации об авторе. Вместо того, чтобы заставлять авторов вставлять разделы About the author (Об авторе) и Abstract (Аннотация) в каждую статью, можно позволить XSL создать этот текст. Кроме того, XSL позволяет менять порядок представления тэгов. Если вы внимательно изучите необработанные XML-данные наших статей, то обнаружите, что имя автора в них предшествует фотографии, в то время как при выводе статьи фотография расположена перед именем автора. Это простейшее использование XSL, но оно наглядно демонстрирует возможности осуществления значительных структурных изменений при выводе существующих XML-данных. Если бы мы захотели произвести глобальные

изменения в представлении статьи пользователю – например, переместить информацию об авторе в самый конец статьи, нам для этого потребуется только изменить правила стилей XSL. Сам текст XML остался бы нетронутым. Действительно, поскольку правила XSL применяются к XML во время исполнения, можно обеспечить различные стили вывода, даже запрещая вывод некоторых тэгов в зависимости от контекста, в котором эти тэги обнаружены.

ASP для презентации

Для осуществления вывода технических статей в нашей системе давайте вновь вернемся к Active Server Pages. Реальное создание DHTML будет происходить на стороне клиента, но при этом необходимо и на сервере проделать определенные операции. Страница, возвращаемая клиенту, будет содержать компонент XSL-процессора. Требуется установить два свойства: XSL-файл, который следует применить, и XML-файл, к которому он должен быть применен. XSL-файл один для всех статей, но чтобы получить имя желаемого XML-файла из строки запроса и вставить его в создаваемую страницу клиента, потребуется небольшой участок кода в ASP. Вспомним, что эта страница, `ViewPaper.asp`, запрашивается с URL `ViewPaper.asp?XML_Filename`. Ниже дан исходный код страницы ASP:

```
    <script FOR="window" EVENT="onload">
        var xslHTML = XSLControl.htmlText;
        document.all.item("xslTarget").innerHTML = xslHTML;
    </script>

    <body>
        <object ID="XSLControl" CLASSID ="CLSID:2BD0D2F2-52EC-11D1-8C69-
0E16BC000000"
        CODEBASE="http://www.microsoft.com/xml/xsl/msxsl.cab" STYLE="display:none">
        <param NAME="documentURL"
            VALUE="<%=Request.ServerVariables("QUERY_STRING") %>">
        <param NAME="styleURL" VALUE="papers.xsl">
        </object>

        <div id="xslTarget"></div>

    </body>
</html>
```

Эта часть кода распознает тэг `<object>` из исходной клиентской страницы поиска:

```
<object ID="XSLControl" CLASSID="CLSID:2BD0D2F2-52EC-11D1-8C69-0E16BC000000"
    CODEBASE="http://www.microsoft.com/xmlxsl/msxsl.cab" STYLE="display:none">
```

Индентификатор `ID` выдает имя компонента, которое может быть использовано при написание скриптов. `CLASSID` отождествляет элемент управления COM с операционной системой, позволяя Internet Explorer найти правильную версию. Атрибут `CODEBASE` просто сообщает браузеру, где находится элемент управления, если он не установлен на системе клиента. Атрибут `STYLE="display:none"` делает невидимым элемент управления на странице. Единственный код, исполняемый на сервере, – это вызов `Request.ServerVariables()`:

```
<param NAME="documentURL" VALUE="<%=Request.ServerVariables("QUERY_STRING") %>">
```

В отличие от предыдущих кодов, расположенных на стороне сервера, в этом случае будет использован синтаксис `<%code%>` для внедрения указанной строки в HTML-данные, которые будут возвращены клиенту. Знак равенства приказывает ASP выводить возвращаемое значение вызова метода в поток HTML, возвращаемый клиенту. Следовательно, если пользователь запросит XML-файл `robot.xml` и впоследствии просмотрит на браузере исходный код HTML, он увидит:

```
<param NAME="documentURL" VALUE="robot.xml">
```

Как работает XSL-процессор

XSL-процессор – это компонент JavaScript. На него ложится основная нагрузка при воспроизведении статьи в приятном для глаз виде. Можно оценить объем выполняемой им работы, если учесть, что прежде чем заняться задачей сопоставления XSL-правил с тэгами в дереве разбора, этот процессор должен выполнить все задачи XML-анализатора.

Совет

Компонент XSL-процессора – это предварительный образец от Microsoft. Поэтому устойчивость его работы и возможности варьируются в зависимости от версии Internet Explorer. В описываемом приложении был использован Internet Explorer 4.0. В момент подготовки книги уже был реализован первый рабочий проект спецификации XSL. Он вносит существенные изменения в синтаксис, используемый в этой главе. Достоверно известно, что в будущем корпорация Microsoft намерена поддерживать новый синтаксис (описанный в главе 8) в своих XSL-процессорах, но очевидно, что в настоящее время представлять рабочие правила таблиц стилей можно только в старом синтаксисе. В любом случае изменения не затрагивают природу XSL: правила применяются к тэгам XML в некотором контексте, меняется только синтаксис. Если Microsoft решит изменить компонент так, что он будет отражать модификации в XSL, нам придется заменить компонент более свежей версией.

Помещение форматированного HTML на страницу

Когда сформированная страница прибывает к клиенту и там загружается, происходят интересные события. На странице создается экземпляр XSL-процессора. Его первая задача – извлечь XML- и XSL-файлы, ссылки на которые имеются в его свойствах. Решив ее, процессор применяет правила из XSL-файла к тэгам из XML-файла и выполняет внутреннюю генерацию HTML. Все это происходит еще до того, как браузер закончит загружать страницу. После того как страница полностью загружена, браузер вызывает событие `onload` (при загрузке). Это приводит к запуску следующей программы обработки события:

```
<script FOR="window" EVENT="onload">
   var xslHTML = XSLControl.htmlText;
   document.all.item("xslTarget").innerHTML = xslHTML;
```

```
</script>
```

Свойство `htmlText` процессора – это строка, содержащая HTML-элементы, созданная путем применения XSL-правил к тэгам XML. Следующая строка кода просто устанавливает, что содержимое области `<DIV>` на этой странице, которому присвоено имя `xslTarget`, соответствует упомянутой строке HTML. Все, что пользователь видит на странице, находится в этой строке и создано XSL-процессором.

Соединение тэгов XML и правил XSL

Ранее уже было сказано, что XSL-процессор применяет правила стилей к тэгам XML. Что это означает конкретно? Файл XSL состоит из совокупности правил, которые должны быть сопоставлены с тэгами, появляющимися в тексте XML-документа. Этот процесс сопоставления позволяет проводить сложное контекстно-чувствительное применение правил стилей; в нашем же примере все достаточно просто.

Совет *Помните, что эти правила таблиц стилей написаны в старом синтаксисе, для того чтобы мы могли работать с существующим XSL-процессором от Microsoft. Сейчас я лишь хочу дать вам почувствовать природу XSL и завершить пример красивым визуальным представлением. Новый синтаксис очень сильно отличается от прежнего, однако и при новом подходе все равно подразумевается та же последовательность действий: согласовать контекст тэгов XML, затем обеспечить заданный вывод данных. Язык XSL подробно описан в главе 8.*

Одно из этих правил определено как *корневое правило* (root rule), с него и начинается обработка:

```
<xsl>
<rule>
   <root/>
   <HTML>
       <SPAN>
           <select-elements>
               <target-element type="paper-title"/>
           </select-elements>
           <BR/>
           <select-elements>
               <target-element type="author"/>
           </select-elements>
           <BR/>
           <select-elements>
               <target-element type="document-body"/>
           </select-elements>
       </SPAN>
   </HTML>
</rule>
```

Это означает, что мы выбрали для вывода на экран название, блок сведений

об авторе и тело статьи (тэг <select-elements> в правиле). Порядок именно таков! Тэги XML, подлежащие сопоставлению, указаны как атрибуты type тэгов <select-elements> в правилах XSL. Ранее мы предлагали переместить блок сведений об авторе в конец выведенного документа. Это произойдет, если переставить тэги для "author" и "document-body". Однако тэг <author> содержит внутри себя другие тэги, так что процессор будет продолжать пытаться сопоставлять правила и тэги в рамках тэга <author>.

Мы решили выводить постоянный подзаголовок About the author белыми буквами на темной полосе. Далее мы хотим создать HTML, чтобы включить в документ фотографию автора (с меняющимся текстом), а за ней имя автора и его биографию. Учтите, что мы изменяем порядок расположения фотографии и имени автора по сравнению с тем, как они были расположены в XML-файле.

```
<rule>
   <target-element type="author"/>
   <DIV background-color="teal"
       color="white"
       font-family="Verdana,arial,helvetica,sans-serif"
       font-size="12pt" padding="4px"> About the author </DIV>
   <SPAN font-family="Verdana, arial, helvetica, sans-serif">
       <select-elements>
           <target-element type="picture"/>
       </select-elements>

       <select-elements>
           <target-element type="name"/>
       </select-elements>
       <BR><BR/>
       <select-elements>
           <target-element type="cv"/>
       </select-elements>
   </SPAN>
</rule>

<rule>
   <target-element type="picture"/>
   <SPAN>
       <IMG SRC="=text" ALT="Author's likeness"/>
   </SPAN>
</rule>

<rule>
   <target-element type="name"/>
   <SPAN font-size="12pt">
   <children/>
   </SPAN>
</rule>

<rule>
   <target-element type="cv"/>
```

```
    <DIV background-color="#EEEEEE" color="black" font-size="10pt">
        <children/>
    </DIV>
</rule>
```

Все, что расположено внутри тэга `<select-elements>` и следующего тэга `<target-element/>` – это HTML, который мы хотели получить. В элементе `"picture"` мы используем директиву `"=text"`, чтобы выполнить контрольное считывание значения тэга:

```
<rule>
    <target-element type="picture"/>
    <SPAN>
        <IMG SRC="=text" ALT="Author's likeness"/>
    </SPAN>
</rule>
```

Перед работой с телом документа проведем аналогичное форматирование названия статьи и аннотации.

```
<rule>
    <target-element type="paper-title"/>
    <SPAN font-family="Arial, helvetica, sans-serif" font-size="14pt"
    _align="left">
        <children/>
    </SPAN>
    <BR/>
</rule>

<rule>
    <target-element type="document-body"/>
        <select-elements>
            <target-element type="abstract"/>
        </select-elements>
        <BR><BR/>
        <select-elements>
            <target-element type="paper-text"/>
        </select-elements>
</rule>

<rule>
    <target-element type="abstract"/>
    <DIV background-color="teal" color="white" font
    _family="Verdana,arial,helvetica,sans-serif" font-size="12pt" padding="4px">
    _Abstract </DIV>
    <DIV id="abstract" font-family="Verdana, arial, helvetica, sans-serif"
    _font-size="10pt" color="black" background-color="#EEEEEE">
        <children/>
    </DIV>
</rule>

<rule>
```

Пространства имен, метаданные и будущие приложения

Вернемся к тому моменту, когда мы работали над формированием правильного XML на стороне клиента. Тогда было отмечено, что возможность получения DTD для проверки состоятельности XML могла бы оказаться желательной. Но ведь мы как раз использовали XML в надежде облегчить взаимодействие клиентов и серверов без предварительной координации. Как же выявить правильную семантику приложения во время транзакции? Два проводимых в настоящее время эксперимента в сообществе XML, а именно *пространства имен* (namespaces) и *XML-данные* (XML-Data), помогут ответить на этот вопрос.

Пространство имен XML

Архитекторам XML скоро стало ясно, что документы могут быть сформированы с использованием схем, поступающих из различных источников, при этом распространенные имена тэгов могут вступать в конфликты. Мы в настоящем сборнике используем типовой термин `paper`. Маловероятно, чтобы вся группа авторов оказалась в Web первыми разработчиками, которые изобрели подобное название для тэга XML. Вот почему клиенты должны знать, какое именно определение элемента следует использовать. Пространства имен – это средство для обозначения источника определения, которое применяется к отдельному тэгу. В документ вставляются элементы, устанавливающие имя вашего пространства имен. Атрибут элемента обеспечивает URL для источника схемы. Тэги, чьи имена могут конфликтовать с другими, даются в имени URL как префикс. В настоящее время синтаксис, который мы могли бы использовать в будущих реализациях нашей системы, выглядит так:

```
<PaperResponse xmlns:TechPapers="http://www.mypapers.org/schema/>
...
</PaperResponse>
```

Совет *Пространства имен – это пока еще рабочий проект консорциума W3C. В ближайшем будущем синтаксис и возможности этого механизма, безусловно, могут по мере развития меняться.*

Ясно, что это шаг в правильном направлении. Пространства имен позволят запрашивать надежные источники семантики приложения для некоторой совокупности тэгов XML. Наш гипотетический программный агент мог бы усваивать информацию, выраженную в наборе тэгов XML, идентифицированных пространством имен. К сожалению, в этом вопросе есть небольшая проблема. DTD записаны в ином синтаксисе, чем XML. Означает ли это, что каждый клиент, будь он агентом, браузером или вспомогательной системой промежуточного ПО, обязан поддерживать два анализатора: один для синтаксиса XML и второй для синтаксиса DTD? Другая разработка, XML-данные, находящаяся на еще более ранней стадии стандартизации, может разрешить проблему, при этом подобный механизм способен открыть новые, более широкие возможности обработки информации.

XML-данные

XML-данные – это словарь XML для определения схем метаданных. Схемы могут быть синтаксическими, как в самом XML, или концептуальными, как в моделях объектов или реляционных базах данных. Таким образом, разработка XML-данных открывает способ определения произвольной конструкции для семантики приложения в формате XML. То есть мы возвращаемся к поддержке одного анализатора, предполагающего, что все верифицирующие XML-анализаторы обладают неявным знанием XML-данных.

Совет *XML-данные – пока еще только сообщение, представленное в консорциум W3C. Это означает, что консорциум подтвердил получение сообщения, но не высказал ни одобрения, ни обещания выделить ресурсы для его оценки. Хотя некоторый синтаксис описания метаданных, выраженный в XML, скорее всего станет членом семьи XML, не ожидайте, что в течение ближайшего времени вы сможете строить реальные программы, основываясь на возможностях XML-данных.*

Если XML-данные войдут в набор инструментов разработчика приложений на сервере, перед ними откроется еще одна возможность. В нашей системе существовали два типа данных, которые мог получить пользователь: краткая информация о содержании технической статьи и полный ее текст. Данные, хранящиеся в базе данных, были спрятаны от пользователя. Хотя подобная инкапсуляция является хорошей практикой программирования, желательно иметь возможность описывать дополнительную информацию, которую мог бы получить пользователь. Мы могли бы задействовать XML-данные, чтобы организовывать некоторое подмножество схемы базы данных. Тогда нетрудно было бы сообщить клиенту, что доступна некая дополнительная информация, например об авторе.

Итак, разрушаются последние признаки жесткого связывания исполняемого кода и семантики данных. Будущие автоматизированные клиенты будут общаться с незнакомыми им серверами, получая схемы словаря того XML, на котором «говорят» серверы. Они смогут узнать, что содержится на сервере. Если это будет соответствовать целям их поиска и их возможностям, они сформируют правильную XML-транзакцию и обработают ответ, полученный в виде XML.

Заключение

В этой главе было показано, как XML может быть использован для облегчения взаимодействия между клиентом и сервером. Конкретно, мы исследовали основы следующих задач:

- программное построение правильных XML-данных;
- передача XML-документа от клиента серверу с использованием одного из нескольких методов обработки запроса;
- анализ XML-кода при помощи реализации объектной модели документа (DOM) для XML средствами Microsoft ActiveX;

- преобразование XML- в DHTML-элементы для вывода на экран;
- воспроизведение XML-документа при помощи предварительного образца XSL-процессора в виде компонента ActiveX.

Мы также выяснили, почему использование XML предпочтительнее по сравнению с более традиционными способами. В противоположность прежним механизмам, XML является открытым самоописывающимся форматом, выраженным в данных текстового типа. Подобный формат поддерживается любой возможной платформой. В настоящее время это обстоятельство позволяет быть терпимым к незначительным ошибкам в реализации описания типа документа (DTD), при условии, что используется скорее просто правильный XML, чем XML, формально состоятельный по отношению к DTD. По ходу нашего разбора мы создали сервер, работа которого была продемонстрирована во взаимодействии с пользователем-человеком, однако тот же аппарат может быть использован программными агентами без всяких изменений кода на сервере. Это полный отход от принципов большинства серверных приложений, написанных с использованием традиционных методов.

В заключение мы заглянули в будущее и бросили взгляд на перспективы, которые язык XML открывает перед приложениями, распределенными по всему Internet. Сильные стороны XML, при их объединении с несколькими новыми инициативами, дадут возможность создавать мощные системы нетрадиционных клиентов, общающихся с серверами, нейтральными к типу клиента. Это сотрудничество ad hoc (по случаю) будет прямо во время общения выявлять степень способностей сервера выполнить запрос пользователя.

В этом случае XML из скромного текстового формата обещает превратиться в по-настоящему мощный и перспективный инструмент.

Глава 11. Учебный пример «Туристический маклер»

В этой главе мы займемся архитектурой XML-приложения. Сначала будут разобраны несколько ключевых моментов, составляющих фундамент данной темы, например, почему XML обеспечивает создание таких приложений, которые ранее было просто невозможно построить. Затем приступим к решению конкретной задачи, в ходе которого обсудим архитектуру, необходимую для получения нужного результата. В конце будет детально рассмотрен каждый компонент кода. Обсуждения предлагаем начать со следующих вопросов:

- основные возможности XML;
- как XML используется в трехуровневой архитектуре;
- последовательность действий при разработке XML DTD;
- отображение базы данных в виде XML-документа;
- построение XML-сервера с помощью инструментов от Microsoft.

Любое приложение строится на основе трехуровневой архитектуры, включающей *службы данных* (Data Services), *бизнес-службы* (Business Services) и *службы пользователя* (User Services). Бизнес-службы действуют подобно агентам, которые, в соответствии с запросом пользователя, извлекают данные из различных источников и собирают их в единый пакет. Службы пользователя принимают информацию от бизнес-служб и форматируют ее для представления пользователю. В завершение главы будут рассмотрены следующие вопросы:

- как использовать объектную модель документа (DOM) для обработки XML-документа;
- как создать новый XML-документ из совокупности других XML-документов;
- как с помощью VBScript использовать процессор Microsoft MSXSL на Active Server Pages;
- как написать таблицу стилей XSL для форматирования XML-документа на сервере.

Изложенный на этих страницах материал потребует знания VBScript, Active Server Pages и некоторого представления о базах данных, особенно SQL. Доступ к базе данных осуществляется при помощи ActiveX Data Objects (ADO), так что полученные по этому вопросу сведения должны вам помочь. И наконец, транспортным протоколом для приложения будет HTTP, так что важно иметь понятие и об URL.

Приложение «Туристический маклер»

Сценарий, в общих чертах, таков: вы хотите провести отпуск на самом подходящем, с вашей точки зрения, курорте. Не исключаются круизы, путешествия и любой другой вид развлечений. При этом вы сразу ограничиваете задачу:

- суммой денег, которую вы готовы потратить;
- днем отъезда;
- страной, в которую вы решили отправиться;
- перечнем услуг, предоставляемых в том или ином месте.

Но этого мало, не плохо также, если и с погодой повезет. Самый изысканный сервис и масса предлагаемых развлечений ни к чему, если вы попадете на курорт в штормовой сезон.

Итак, вы решили отыскать идеальное по вашим представлениям местечко через World Wide Web. Вы запускаете поисковую машину и начинаете просматривать Web-сайты. Вскоре до вас доходит, что без бумаги и карандаша вам не обойтись. Сравнить, а тем более оценить сведения, относящиеся к тому или иному пункту, фирме или маршруту, без специально вычерченной таблицы просто немыслимо. Еще через пару часов вас начинает донимать простенькая мыслишка: «Неужели нельзя придумать способ получше?!».

Решение

Вскоре ваш безмолвный вопль услышан, и вдруг перед вами оказывается «Туристический маклер»! Вот и ответ!.. «Туристический маклер» – это специальный Web-сайт (к сожалению, в настоящее время не существующий), который автоматически, имея в виду ваши пожелания, найдет подходящий вариант. Ваша задача суживается до перечисления указанных выше требований и описания устраивающего вас прогноза погоды (желаемая температура, влажность, ветер, вероятность осадков и т.п.). Вводится соответствующий код, и на экране тотчас высвечивается список курортов, охватывающий предложения разных турбюро и агентств.

Конечно, и в настоящее время на World Wide Web существуют подобные маклерские Web-сайты. Проблема заключается в том, что большинство подобных документов требует от разработчиков, их поддерживающих, непомерно огромной работы по вводу актуальных и точных данных, поступающих от различных агентств. Подобный сайт сам по себе ничего не продает, он только собирает информацию от многих турбюро и объединяет ее. При этом формирование единого списка осуществляется вручную. Web-мастер должен получить информацию от каждого турбюро и свести все эти сведения воедино. Было бы куда удобнее, если бы турагенства со своей стороны публиковали информацию в каком-то хорошо известном формате, чтобы ее можно было бы автоматически и регулярно собирать. Такой подход исключает ручной ввод и процесс объединения информации и, безусловно, способен обеспечить актуальность данных. Существующие автоматизированные Web-сайты обычно используют сложные технологии (например, LDAP и EDI). В свою очередь, XML обеспечивает стандартную схему данных и не предъявляет высоких требований к авторам. Рассмотрим архитектуру ныне существующих систем (рис. 1.1).

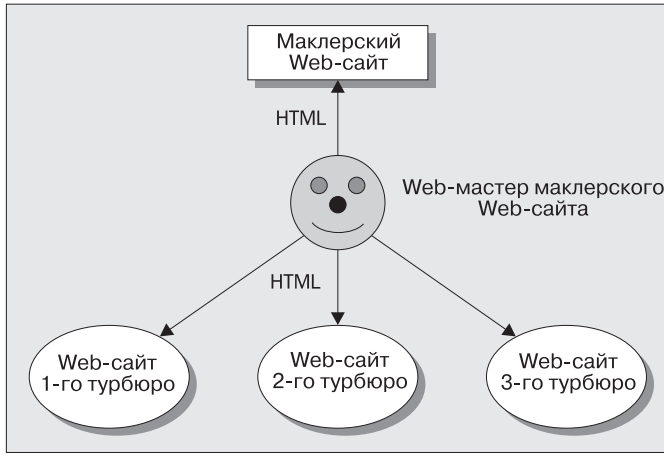


Рис. 11.1. Схема работы существующих маклерских Web-сайтов

Альтернативная система, основанная на XML, имела бы аналогичную структуру, но работа по сведению данных выполнялась бы не вручную, а с помощью особого приложения (рис. 1.2).

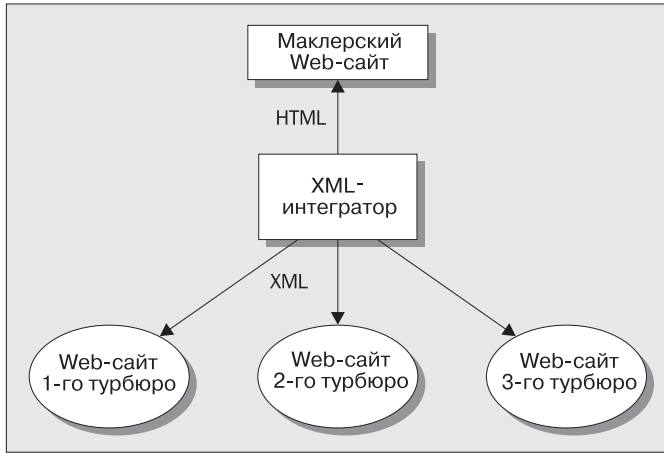


Рис. 11.2. Схема работы автоматизированного маклерского Web-сайта

Архитектура

Многие организации внедряют решения, построенные на логической взаимосвязи служб. Каждая служба как поставляет, так и потребляет данные, а также обеспечивает выполнение функциональных обязанностей других отделов. Они могут быть распределены по физическому пространству или по разным операционным системам. Часто логически взаимосвязанные сети организуются на принципах трехуровневой архитектуры. Эта модель очень хорошо согласуется с Web-приложениями, использующими XML. Теперь самое время обсудить настраиваемую архитектурную конструкцию, состоящую из трех уровней, а затем расмотреть вопросы ее согласования с нашим Web-приложением.

Трехуровневая архитектура

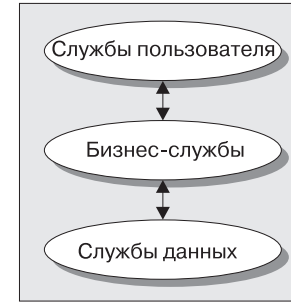


Рис. 11.3. Трехуровневая архитектура приложений

Подобная структура, как уже было сказано, логически делится на три отдельные области: службы данных, бизнес-службы и службы пользователя (рис 11.3). Все вместе они образуют взаимосвязанное, гибкое и расширяемое приложение. Отметим, что приложение может задействовать как все подобные подразделения, так и только некоторые из них. Например, службы пользователя для банковских приложений могут задействовать две различные службы данных: одну службу данных для оплаты выставленных счетов и другую – для операций с банковскими счетами. Отметим, что разработчику этой системы следует позаботиться о механизме транзакций, чтобы обеспечить согласованность между отдельными службами данных.

Службы данных

Службы данных отвечают за поиск и хранение информации. Эти службы обеспечивают абстракцию данных для бизнес-служб, которым в подобном случае не требуется знать, где расположены данные, как это расположение реализовано и как осуществляется доступ к данным. Службы данных должны обеспечивать механизм поддержания целостности всего объема сведений, например такого, как менеджмент транзакций.

Бизнес-службы

Бизнес-службы отвечают за исполнение конкретных бизнес-правил. Они взаимодействуют со службами данных, запрашивая и сохраняя полученные сведения. К полученным данным могут быть применены различные операции, а также вычисления и проверки на правильность. Бизнес-службы, применяя особые правила, трансформируют данные в информацию. Бизнес-службы не обладают средствами визуального отображения.

Службы пользователя

Службы пользователя – это визуальная часть информационного приложения. Они форматируют и отображают данные. Они также предоставляют механизмы, при помощи которых пользователь может управлять данными. Правильно определенные службы пользователя принимают во внимание виды деятельности, в которые вовлечен пользователь, и стили взаимодействия, им ожидаемые.

Трехуровневая архитектура, использующая XML

XML великолепно подходит для трехуровневой архитектуры. XML, образно говоря, напоминает что-то вроде склеивающей основы между тремя уровнями служб. Этот язык позволяет применять различные пользовательские интерфейсы для работы с одним и тем же приложением и позволяет приложениям легко взаимодействовать с различными источникми данных.

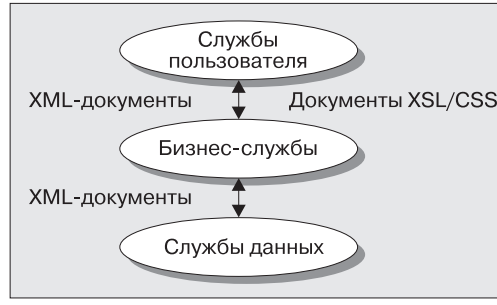


Рис. 11.4. Трехуровневая архитектура приложения с использованием XML

Службы данных: создание XML-документа

Источник данных для приложения находится на уровне данных. Пользователя этих служб не интересует, как и где хранится информация. Поскольку XML действует как универсальный формат для различных хранилищ данных, он представляет собой идеальный механизм для служб данных. Язык XML обеспечивает доступность уровня служб данных, позволяя публиковать сведения в общеизвестном формате. Теперь любой клиент может пользоваться данными, не испытывая необходимости применять специализированные средства API для каждого типа информации, находящейся в источнике данных. Данные выходят из служб данных в структурированном виде, а именно, в виде XML. Примером службы данных, выполненной с использованием XML, является библиотека, предоставляющая доступ к базе данных хранимых книг через XML-документы.

Бизнес-службы: интеграция данных

Бизнес-службы в XML-приложении действуют подобно агенту пользователя, который принимает запрос, решает, какая информация требуется и как ее извлечь. После получения информации агент конвертирует ее в форму, удобную для пользователя. Например, запрос пользователя может преобразоваться в запрос десяти XML-документов из различных источников данных. На уровне бизнес-служб документы обрабатываются, и в результате формируется один XML-документ для представления пользователю. Бизнес-службы проводят автоматизированный поиск и фильтрацию, а также объединение различной информации в XML-документ.

Службы пользователя: HTML-презентация

Службы пользователя готовят информацию для ее использования человеком. Преимущество использования XML заключается в том, что форма представления данных может быть легко изменена без какого-либо влияния на сами данные. Идеальным выбором для представления информации является HTML, поскольку HTML является языком представления данных во всех Web-браузерах. Существует несколько способов преобразования XML-данных в HTML-данные. Документ может быть отформатирован на Web-сервере и послан клиенту в виде HTML, либо

его можно непосредственно послать браузеру клиента вместе с инструкциями по форматированию. Теоретически инструкциями по форматированию являются либо каскадные таблицы стилей (CSS), либо таблицы стилей XSL. Рассмотрим возможные варианты более подробно.

Форматирование на сервере

При форматировании документа для презентации на сервере мы защищены от различий в реализации XSL или каскадных таблиц стилей (CSS) в браузерах, что, конечно, является бесспорным преимуществом подобных механизмов. Недостатком такого подхода является дополнительный объем работы, выполняемый на сервере, что увеличивает время ответа клиенту. Это нежелательно, поскольку одной из главных целей подобной обработки является стремление переложить на клиента часть операций, касающихся ответов на часть запроса. При форматировании на сервере Web-сервер имеет встроенный XSL- или CSS-процессор для построения HTML-документа, передаваемого клиенту.

Форматирование на стороне клиента

Отправка XML-документа с инструкцией по его форматированию перекладывает нагрузку с сервера на компьютер клиента. В результате повышается эффективность системы в целом. Однако поскольку каждый браузер по-разному обрабатывает XSL и CSS (если он их вообще поддерживает), результаты могут оказаться непредсказуемыми. Форматирование на стороне клиента хорошо работает в Intranet, потому что в этом случае можно проследить за типом и версией клиентских браузеров.

Форматирование с использованием CSS

Спецификация CSS 1 доступна с декабря 1996 года, и потому уже могла быть использованна программистами. Частично CSS 1 уже работает с HTML в браузерах IE 3.0, IE 4.0 и Navigator 4.0. Но ни в одном из ныне существующих Web-браузеров CSS не работает с XML. В IE 5.0 будет реализована некоторая поддержка использования CSS и XML. Более подробную информацию о таблицах стилей вы найдете в книге Professional Style Sheets for HTML and XML, Wrox Press (ISBN 1-861001-657).

Форматирование с использованием XSL

Язык XSL сейчас находится в стадии разработки. Первый проект был реализован 18 августа 1998 года. Это означает, что на данном этапе разработка реальных систем с использованием XSL хорошей идеей считаться не может, поскольку спецификация XSL, скорее всего, еще не раз претерпит значительные изменения. К тому же в основных Web-браузерах пока еще нет собственной поддержки XSL. Преимущество трехуровневой системы заключается в том, что можно сначала реализовать службу пользователя при помощи CSS 1, а когда XSL станет более устойчивым, перейти на него, причем подобный процесс не повлияет на остальные части системы.

Использование множественных решений

На практике выход только один – реализовать множественный подход. Подобные решения имеют место уже в настоящее время, когда разработчики

создают различные версии собственных Web-сайтов для различных браузеров. Более того, некоторые программисты также создают версии таких Web-сайтов с использованием и без использования Java и фреймового подхода. В таких случаях либо пользователю предоставляется выбор типа Web-сервера, либо Web-сервер автоматически определяет, какой браузер осуществляет запрос, и передает ему соответствующие версии документов. Этот подход явно требует больших усилий, выигрыш же заключается в том, что можно использовать усовершенствованные возможности браузеров, чтобы переложить часть нагрузки на компьютер клиента и вдобавок добиться более привлекательного вида Web-страниц.

Например, в случае с XML можно установить следующие правила для приложения:

- если браузером является IE 4.0, то следует вернуть XML-документ с каскадными таблицами стилей (CSS);
- в противном случае следует отформатировать XML-документ на сервере как HTML, используя XSL-процессор, и послать его клиенту *или*
- следует отформатировать документ как HTML на сервере, используя CSS, и послать его клиенту.

С точки зрения самого процесса предоставления данных, содержание (сам XML-документ) остается одним и тем же вне зависимости от типа браузера. Выбор, таким образом, остается за разработчиком. Он решает, поддерживать ли для применения стилей одновременно CSS и XSL или использовать только CSS.

Теперь можно сказать, что мы почти готовы приступить к детальному построению «Туристического маклера». Схема на рис. 11.5 показывает соотношение концептуальной трехуровневой архитектуры с реально выстраиваемыми компонентами. Даны также имена zip-файлов, содеращих все необходимые исходные коды.

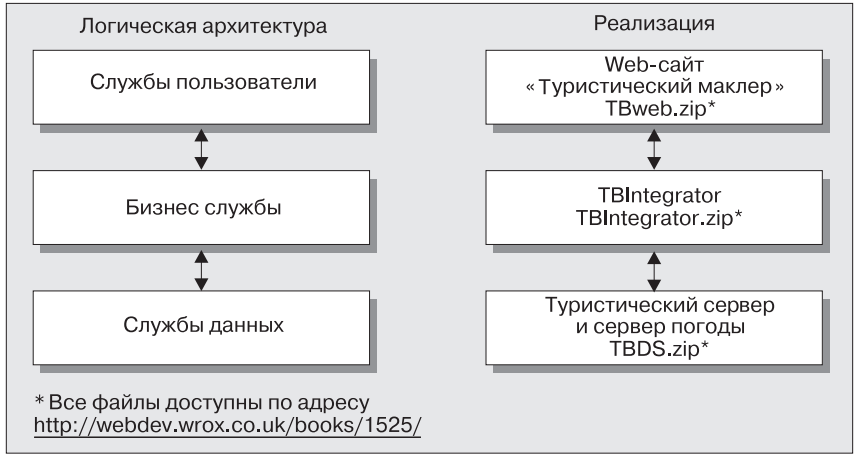


Рис. 11.5. Трехуровневая архитектура и компоненты приложения «Туристический маклер»

Службы данных приложения

XML-сервер «Туристического маклера» представляет собой службы данных приложения «Туристический маклер». Он отвечает за поддержку сведений, необходимых для приложения, а также за их вывод в правильном формате, то есть в виде XML-документа.

Данные включают сведения о том, какие путевки имеются в наличии, на какой срок, с какого числа, какова их стоимость. Нас также интересует предварительный прогноз, и службы данных имеют возможность предоставить нам информацию о погоде в разных местах в разное время года.

Обращаясь к приложению «Туристический маклер», мы запрашиваем службы данных только о свободных или «горящих» путевках. В реальных системах другие приложения, например по бронированию мест, тоже могли бы использовать службы данных. Информацию о погоде можно передавать другим независимым приложениям, связанным, например, с сельским хозяйством или диспетчерской службе, координирующей полеты самолетов. Как вы, наверное, знаете, подобная информация хранится в базах данных.

Базы данных

Наше приложение будет использовать две базы данных.

База данных о путевках

База данных, охватывающая сведения о путевках, содержит всю информацию, которая имеется в распоряжении турбюро и касается прежде всего наличия свободных мест. Бюро или агентства могли бы иметь свою собственную базу данных. Хранимая информация включает: описание условий путешествия или курорта, сезон, на который они запланированы или предпочтительное время посещения и наличие свободных мест для определенного дня заезда.

Структура

База данных путевок разделена на три таблицы: `Packages` (Путевки), `Price` (Цена), `Availability` (Наличие). Структура базы данных представлена на схеме (рис. 11.6).

Таблица `Packages` описывает путешествие, которое можно выбрать. В ней представлены `Country` (Страна), `City` (Город) и `ResortName` (Название курорта), которые дают информацию о расположении курорта. Об уровне обслуживания сообщают поля `Rating` (Оценка), `WaterSports` (Водный спорт), `Meals` (Еда), `Drinks` (Напитки). Цены на отдых на курорте варьируются в зависимости от времени года, поэтому они размещены в отдельной таблице, названной `Prices`. И наконец, информация о местных датах заезда и о наличии свободных мест для этих дат размещена в таблице `Availability`.

Реализация

В этом примере база данных будет создана в Microsoft Access. Обращаться к ней будем через *ODBC*, так что вы можете, если хотите, использовать эту схему для построения своей собственной базы данных. ODBC – это акроним *открытого интерфейса подключения к базам данных* (Open Database Connectivity). ODBC обеспечивает стандартный API вне зависимости от системы управления базой данных. База данных доступна на http://webdev.wrox.co.uk/books/1525.

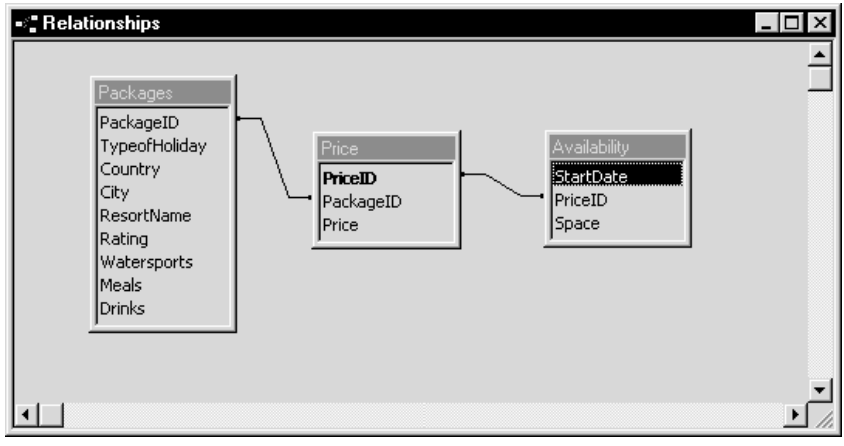


Рис. 11.6. Структура базы данных по путевкам

Вы можете запросить эту базу данных при помощи своего браузера и затем сохранить ее на диске, когда появится соответствующее диалоговое окно. Когда вы это проделаете, вам будет необходимо привязать ее к ODBC.

Дважды щелкните по пиктограмме ODBC на **Control panel** (Панель управления) вашей системы Windows. Затем щелкните по закладке свойства **System DSN** (Системный DSN). Поскольку Web-сервер будет обращаться к базе данных, она должна быть системным DSN. Затем щелкните по **Add** (Добавить) и выберите **Microsoft Access Driver** (Драйвер Microsoft Access). Наконец заполните диалоговое окно, как показано на рис. 11.7. Учтите, что когда вы выбираете базу данных, следует указать пть к сохраненной на вашем диске базе данных.

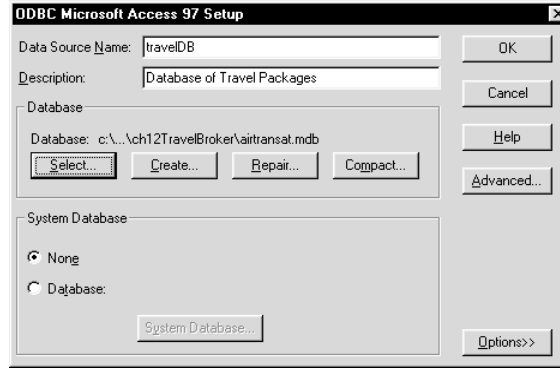


Рис. 11.7. Привязка базы данных с информацией о путевках к ODBC

Теперь вы легко сможете обращаться к базе данных через ODBC.

База данных с информацией о погоде

Эта база данных содержит сведения о погоде в различные сезоны для разных районов земного шара. Она описывает метеорологическую ситуацию, обычную для конкретного места в конкретном месяце. В базу данных включена также информация о диапазоне температур, средней скорости ветра и среднем количестве осадков.

Структура

В базе данных с информацией о погоде имеются две таблицы: `Location` (Место) и `Weather` (Погода) Структура базы данных показана на рис. 11.8.

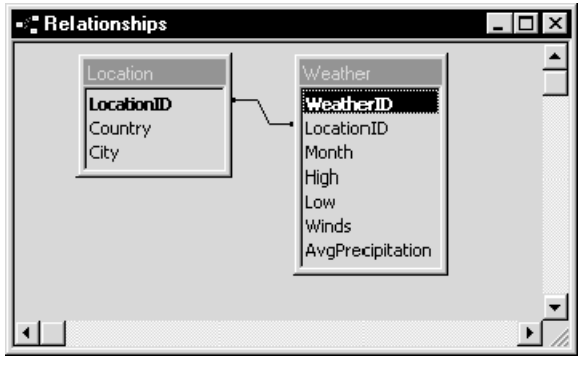


Рис. 11.8. Структура базы данных с информацией о погоде

Таблица `Location` содержит перечнь городов и стран, для которых имеется информация о погоде. Таблица `Weather` связана с таблицей `Location` и содержит подробные метеорологические сведения о погоде для каждого месяца в году. Эта информация включает верхний и нижний предел температуры (по Цельсию), скорость ветра (в км/ч) и среднее количество осадков за месяц.

Реализация базы данных

Базу данных Microsoft Access для этого примера можно получить с http://webdev.wrox.co.uk/books/1525. И опять же эта база данных должна быть привязана к ODBC. Повторите те же операции, что вы проделывали для базы данных с информацией о путевках, кроме последнего диалогового окна, которое теперь должно выглядеть так, как показано на рис. 11.9.

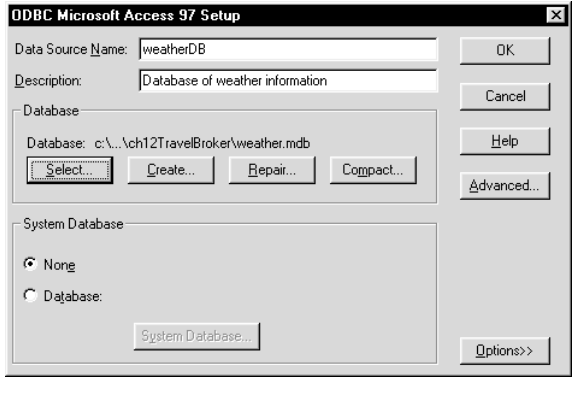


Рис. 11.9. Привязка базы данных с информацией о погоде к ODBC

Для того чтобы добиться цели – в нашем случае вывода правильных (в смысле XML) данных – необходимо описать правила организации этих данных. Правила организации данных публикуются вместе с определением типа документа (DTD).

DTD может быть использовано многими людьми и является основой для интерпретации данных. Поэтому необходимо затратить время и усилия на его создание.

Определения типа документа для XML

Атрибуты или элементы

Основные споры, которые ведутся вокруг конструкции DTD, касаются вопроса, что использовать для представления информации: атрибуты или элементы. Здесь следует заметить, что единственно правильного способа построения DTD не существует. Поскольку у меня уже сложилась привычка работать на основе объектно-ориентированного подхода, я использовал идеи данного способа программирования в качестве руководства при разработке XML. Это завление сводится к тому, что чаще всего, если некоторая информация «имеется», я обозначаю ее элементом. Если же информация «чем-то или каким-то является», я использую атрибут. Например, если бы я описывал автомобиль, то двери, окна, двигатель были бы элементами, потому что автомобиль имеет эти объекты. Однако, цвет марка и год выпуска были бы атрибутами автомобиля.

DTD приложения «Туристический маклер»

Чтобы разработчики знали, как согласовывать свои действия с документом путевок, им необходимы некоторые правила. Ниже приведено определение DTD, очерчивающее эти правила.

```
<!-- DTD для путевок. -->

<!-- Начало описания элементов. -->
<!ELEMENT travelpackages (country*)>
<!ELEMENT country (city+)>
<!ELEMENT city (resort+)>
<!ELEMENT resort (package+)>
<!ELEMENT package EMPTY>
<!-- Конец описания элементов. -->

<!-- Начало описания атрибутов. -->
<!ATTLIST country name CDATA #REQUIRED>
<!ATTLIST city name CDATA #REQUIRED>
<!ATTLIST resort
   name CDATA #REQUIRED
   rating (1|2|3|4|5) #IMPLIED
   typeofholiday (Beach|Touring) #IMPLIED
   watersports (True|False) #IMPLIED
   meals (True|False) #Implied
   drinks (True|False) #IMPLIED>
<!ATTLIST package
   dateofdep CDATA #REQUIRED
   price CDATA #REQUIRED>
<!-- Конец описания атрибутов. -->
```

Отсюда ясно, что корневым элементом документа является элемент `travelpackages`. Корневой элемент может содержать ноль или больше элементов `country` (страна). Если запрошенный туристический сервер возвращает документ без

элементов country, это означает, что записи не были найдены. Элемент country должен содержать один или больше элементов city. Мы принудительно вводим правило, что если страна возвращена, то в одном из дочерних узлов должна быть дополнительная информация. Элемент city (город) обязан иметь один или больше элементов resort (курорт). Курорт – это конкретное место, куда можно поехать отдыхать. И наконец элемент resort должен содержать один или больше элементов package (путевка). Путевка – это сведения о том, когда можно поехать на курорт, и сколько это будет стоить. Элемент package не может иметь дочерние элементы.

Элементы country и city каждый имеют по одному атрибуту name (имя). Атрибут имени идентифицирует страну и город и поэтому необходим.

Элемент resort также требует атрибут name. В дополнение к атрибуту name, элемент resort имеет подразумеваемые атрибуты, которые описывают его. Поскольку эти атрибуты подразумеваемые, они могут быть как перечислены, так и не перечислены. Ниже перечислены необязательные атрибуты:

- rating (оценка) – рейтинг курорта по шкале от 1 до 5;
- typeofholiday (вид отдыха) – классификация видов отдыха. Отдых может быть Beach (Пляжный) или Touring (Экскурсии);
- watersports (водный спорт) – принимает значения True (Истинно) или False (Ложно). Если этот атрибут имеет значение True, то на курорте водные виды спорта представлены;
- meals (еда) – если этот атрибут имеет значение True, стоимость питания включена в путевку;
- drinks (напитки) – если этот атрибут имеет значение True, стоимость напитков включена в путевку.

Отметим, что для атрибутов, принимающих значения true (истинно) или false (ложно), отсутствие атрибута не означает, что данная услуга на курорте не предоставляется. Его отсутствие только показывает, что невозможно определить, предоставляется или нет на курорте данная услуга. Другой способ отображения потребовал бы атрибутов, которые могли бы иметь три состояния: True, False, Unknown (Неизвестно).

Элемент package имеет два обязательных атрибута dateofdep (дата заезда) и price(цена). Дата начала пребывания на курорте представлена элементом dateofdep. Цена путевки для одного человека представлена элементом price.

Ниже приведен пример документа, созданного в соответствии с описанным выше DTD:

```
<?xml version="1.0"?>
<!DOCTYPE travelpackages SYSTEM "travel.dtd">
<travelpackages>
   <country name="Cuba">
       <city name="Cayo Coco">
           <resort name="Club Tryp Cayo Coco" rating ="4" typeofholiday ="Beach"
               watersports ="True" meals ="True" drinks ="True">
               <package dateofdep="5/8/98" price="879"/>
               <package dateofdep="5/1/98" price="879"/>
```

```
            </resort>
        </city>
        <city nam="Varadero">
            <resort name="Sol Club Palmeras" rating ="3" typeofholiday ="Beach"
                watersports ="False" meals ="True" drinks ="False">
                <package dateofdep="5/30/98" price="799"/>
                <package dateofdep="5/23/98" price="889"/>
                <package dateofdep="5/16/98" price="889"/>
            </resort>
        </city>
    </country>
</travelpackages>
```

Здесь наглядно видно, что на этот момент имеется два курорта, соответствующих нашему запросу. Оба расположены на Кубе, один в Кайо-Коко (Cayo Coco), а другой в Варадеро (Varadero). Оба они – пляжные, на одном водные виды спорта входят в услуги, на другом – нет. В стоимость путевок на оба курорта еда включена, а в одну из путевок также входят напитки. И, наконец, имеется набор возможных дат заезда.

DTD погоды

Чтобы разработчики знали, как согласовывать свои действия с документом, несущим информацию о погоде, им необходимы некоторые правила. Ниже приведено определение DTD, описывающее эти правила.

```
<!-- DTD для данных о погоде. -->

<!-- Начало описания элементов. -->
<!ELEMENT weather (country*)>
<!ELEMENT country (city+)>
<!ELEMENT city (month+)>
<!ELEMENT month (avgprecipitation? | winds? | high | low)>
<!ELEMENT avgprecipitation EMPTY>
<!ELEMENT winds EMPTY>
<!ELEMENT high EMPTY>
<!ELEMENT low EMPTY>
<!-- Конец описания элементов. -->

<!-- Начало описания атрибутов. -->

<!ATTLIST country name CDATA #REQUIRED>
<!ATTLIST city name CDATA #REQUIRED>
<!ATTLIST month
    name (january|february|march|april|may|
        june|july|august|september|october|november|december) #REQUIRED>
<!ATTLIST avgprecipitation
    cm CDATA #REQUIRED
    in CDATA #IMPLIED>
<!ATTLIST winds
    km/h CDATA #REQUIRED
    mph CDATA #IMPLIED>
<!ATTLIST high
```

```
        c CDATA #REQUIRED
        f CDATA #IMPLIED>
<!ATTLIST low percent
        c CDATA #REQUIRED
        f CDATA #IMPLIED>
<!-- Конец описания атрибутов. -->
```

Корневым элементом документа является `weather`. Как и в DTD для документа путевок, корневой элемент может иметь ноль или больше элементов `country` (страна). Если элемент `country` существует, он обязан иметь дочерние элементы. Дочерним для элемента `country` является элемент `city` (город). Дочерним для элемента `city` является элемент `month` (месяц). Элемент `month` обязан иметь диапазон температур, поэтому обязательны элементы `high` (верхний предел) и `low` (нижний предел). У нас может быть ноль или больше элементов `avgrecipitation` (среднее количество осадков) и ноль или больше элементов `winds` (скорость ветра). Отметим, что величины могут быть выражены как в метрических единицах, так и в английской системе. Поскольку значения одного типа могут быть вычислены из значений другого типа, значения в метрических единицах обязательны, а в английских – необязательны.

И снова элементы `country` и `city` должны быть идентифицированы, поэтому они имеют обязательный атрибут `name` (имя). Элемент `month` тоже имеет обязательный атрибут `name`. Используя атрибуты при описании погоды, мы допускаем наличие различных систем единиц измерения. Атрибуты для величин в метрических единицах обязательны, в английских – необязательны.

Ниже приведен пример документа, созданного в соответствии с этим DTD:

```
<?xml version="1.0"?>
<!DOCTYPE weather SYSTEM "weather.dtd">
<weather>
   <country name="United States">
       <city name="Fort Lauderdale">
           <month name="April">
               <avgprecipitation CM="8"/>
               <winds kph="40"/>
               <high c="28"/>
               <low c="21"/>
           </month>
           <month name="May">
               <avgprecipitation CM="6"/>
               <winds kph="45"/>
               <high c="30"/>
               <low c="22"/>
           </month>
       </city>
   </country>
</weather>
```

Этот документ отражает состояние погоды в Форт-Лэйдердэйл (Fort Layderdale) в США в апреле и мае. Детальная информация о погоде включает

среднее количество осадков в сантиметрах, скорость ветра в километрах в час и диапазон температур (максимальная и минимальная) в градусах Цельсия.

Реализация при помощи ASP и ADO

Транспортный протокол служб данных реализован через HTTP. В качестве протокола сообщений между уровнями HTTP был выбран вместо других протоколов, таких как DCOM и CORBA, по нескольким причинам. Он, по всей видимости, является самым известным протоколом, поэтому его удобно использовать в обучающих примерах, подобных нашему «Туристическому маклеру». Он прост, недорог, поддерживается различными операционными системами. Правда, одним из его недостатков можно считать необходимость принимать специальные меры, иначе трудно обеспечить безопасность сообщений.

В нашем примере мы будем использовать Microsoft IIS4 с Active Server Pages (ASP), VBScript и ActiveX Data Objects (ADO). Можно пользоваться и другими системами, если они предоставляют такой же интерфейс. Фактически за XML и трехуровневой архитектурой проглядывает идея обеспечить взаимодействие между различными операционными системами и тем самым создать интегрированную, но разнообразную сеть.

Такой способ реализации был выбран по нескольким причинам. XML-документы могут создаваться этой системой без каких-либо специализированных компонентов. За работой VBScript легко проследить, поэтому он хорошо подходит для целей демонстрации. Кроме того, многие знакомы с устройством ASP/VBScript/ADO. Недостаток такого подхода состоит в том, что невозможно получить такую же эффективность, какую дало бы решение, скомпилированное в собственный исполняемый код. Здесь также не проводится проверка созданного XML-документа на состоятельность.

Если вы хотите сразу посмотреть демонстрационный пример, зайдите на http://webdev.wrox.co.uk/books/1525.

Туристический сервер

Туристический сервер реализован в виде ASP с использованием IIS4. Каждое турбюро имеет по своему собственному серверу. Клиент готовит HTTP-запрос в виде URL, осуществляет этот запрос и получает ответ через HTTP. Изучим такой процесс более подробно.

Ввод

Запрос компонуется из имени Web-сервера, имени активной страницы сервера (ASP) и параметров запроса. Параметры содержатся в URL в виде пар имя-значение в строке запроса (имена и описания параметров приведены в табл. 11.1). Конкретно, строка-запрос выглядит так:

```
http://server/travelds.asp?Country=VAL1&City=VAL2&Month=VAL3& Watersports=VAL4&Pri
ce=VAL5&Space=VAL6
```

Необязательность параметра означает, что включение в URL пар имя-значение не обязательно.

Таблица 11.1. Параметры запроса на туристический сервер

server	Имя домена туристического сервера
Country (optional)	Страна, для которой запрашиваются путевки (необязательно)
City (optional)	Город, для которого запрашиваются путевки (необязательно)
Month (optional)	Месяц, для которого запрашиваются путевки (необязательно)
Watersports (optional)	Истина, если в путевку входят водные виды спорта (необязательно)
Price (optional)	Максимальная цена (необязательно)
Space (optional)	Число людей, едущих по путевке (необязательно)
VAL(1-6)	Величины, ограничивающие область поиска совпадений

Вывод

Для вывода создается XML-документ, который соответствует DTD путевки. Он содержит результаты, соответствующие деталям переданного запроса. Если никакие параметры указаны не были, в документе возвращаются все записи. Рассмотрим пример:

```
http://server/travelds.asp?Country=Canada&Month=January
```

В этом запросе будут возвращены все имеющиеся записи об отдыхе в Канаде в январе. Учитывая страну и время года, это будут, скорее всего, лыжные курорты. Если бы вы не знали, какая погода в Канаде зимой, и рассчитывали полежать на пляже, вас ожидал бы неприятный сюрприз. Именно сейчас наступает пора отбора с использованием метеорологического сервера (`weather server`).

Реализация

Ниже дан код ASP для файла `travelds.asp`. Он осуществляет всю необходимую работу от получения запроса до возвращения XML-документа с результатами.

```
<%@ LANGUAGE=VBSCRIPT%>
<% Response.ContentType = "text/plain" %>
<%

' Строим SQL-запрос из переменных формы, переданной в URL.
Dim curCountry
Dim firstCountry
Dim curCity
Dim firstCity
Dim resort
Dim firstResort
Dim doEndTags

firstCountry = True
firstCity = True
firstResort = True
doEndTags = False

SQLStatement = "SELECT * FROM Price tPrice, Packages tPkg, Availability tAvail
WHERE tPkg.PackageId = tPrice.PackageId AND tPrice.PriceID = tAvail.PriceID"

If Len( Request.QueryString( "Month" ) ) > 0 Then
   SQLStatement = SQLStatement + " AND Month(tAvail.StartDate ) = '" +
Request.QueryString( "Month" ) + "'"
End If
```

```
If Len( Request.QueryString( "Country" ) ) > 0) AND Request.QueryString( "Country"
) <> "Any" Then
   SQLStatement = SQLStatement + " AND tPkg.Country = '" + Request.QueryString(
"Country" ) + "'"
End If
If Len( Request.QueryString( "City" ) ) > 0 Then
   SQLStatement = SQLStatement + " AND tPkg.City = '" + Request.QueryString(
"City" ) + "'"
End If
If Len( Request.QueryString( "Watersports" ) ) > 0 Then
   SQLStatement = SQLStatement + " AND tPkg.Watersports = " + Request.QueryString(
"Watersports" )
End If
If Len( Request.QueryString( "Price" ) ) > 0 Then
   SQLStatement = SQLStatement + " AND tPrice.Price <= " + Request.QueryString(
"Price" )
End If
If Len( Request.QueryString( "Space" ) ) > 0 Then
   SQLStatement = SQLStatement + " AND tAvail.Space >= " + Request.QueryString(
"Space" )
End If

SQLStatement = SQLStatement + " ORDER BY tPkg.Country, tPkg.City, tPkg.ResortName"
%>

<?xml version="1.0"?>
<!DOCTYPE travelpackages SYSTEM "travel.dtd">
<travelpackages>
<%
   Set Conn = Server.CreateObject("ADODB.Connection")
   Conn.Open "travelDB"
   Set travelRecs = Conn.Execute(SQLStatement)

   ' Производим цикл по записям, создающим элементы XML.
   Do While Not travelRecs.EOF

       doEndTags = True

       ' Начинаем новый элемент COUNTRY, если находимся в новой стране.
       If curCountry <> travelRecs( "COUNTRY" ) Then
           If firstCountry = False Then
               Response.Write( vbTab + vbTab + vbTab + "</resort>" + vbNewLine )
               Response.Write( vbTab + vbTab + "</city>" + vbNewLine )
               Response.Write( vbTab + "</country>" + vbNewLine )
               firstResort = True
               firstCity = True
           End If
           curCountry = travelRecs( "COUNTRY" )
           Response.Write( vbTab + "<country name=" + Chr(34) + curCountry +
Chr(34)
                   + ">" + vbNewLine )
           firstCountry = False
       End If
```

```
        ' Начинаем новый элемент CITY, если находимся в новом городе.
        If curCity <> travelRecs( "CITY" ) Then
            If firstCity = False Then
                Response.Write( vbTab + vbTab + vbTab + "</resort>" + vbNewLine )
                Response.Write( vbTab + vbTab + "</city>" + vbNewLine )
                firstResort = True
            End If
            curCity = travelRecs( "CITY" )
            Response.Write( vbTab + vbTab + "<city name=" + Chr(34) + curCity +
                    Chr(34) + ">" + vbNewLine )
            firstCity = False
        End If

        ' Начинаем новый элемент RESORT, если находимся на новом курорте.
        If curResort <> travelRecs( "RESORTNAME" ) Then
            If firstResort = False Then
                Response.Write( vbTab + vbTab + "</resort>" + vbNewLine )
            End If
            curResort = travelRecs( "RESORTNAME" )
            Response.Write( vbTab + vbTab + vbTab + "<resort name=" + Chr(34) +
                    curResort + Chr(34) )
            Response.Write( " vendor =" + Chr(34) + "Sunquest" + Chr(34) )
            Response.Write( " rating =" + Chr(34) + CStr(travelRecs( "RATING" ))
                    + Chr(34) )
            Response.Write( " typeofholiday =" + Chr(34) + CStr(travelRecs(
                    "TYPEOFHOLIDAY" )) + Chr(34) )
            Response.Write( " watersports =" + Chr(34) + CStr(travelRecs(
                    "WATERSPORTS" )) + Chr(34) )
            Response.Write( meals =" + Chr(34) + CStr(travelrecs( "MEALS" )) +
                    Chr(34) )
            Response.Write( " drinks =" + Chr(34) + CStr(travelRecs( "DRINKS" ))
                    + Chr(34) + ">" + vbNewLine )
            firstResort = False
        End If

    %>
                <package dateofdep="<%=travelRecs( "STARTDATE" )%>"
                    price="<%=travelRecs( "PRICE" )%>"/>
<%
    travelRecs.MoveNext
    Loop

    ' Добавляем замыкающие тэги, только если действительно есть элементы.
    If doEndTags = True Then
        Response.Write( vbTab + vbTab + vbTab + "</resort>" + vbNewLine )
        Response.Write( vbTab + vbTab + "</city>" + vbNewLine )
        Response.Write( vbTab + "</country>" + vbNewLine )
    End If
%>
</travelpackages>
```

Рассмотрим подробнее этот код. Он формирует SQL-инструкцию, использует ADO для осуществления запроса к базе данных и при помощи VBScript производит сборку XML-документа.

Из SQL-запроса возвращаются все поля из всех таблиц в базе данных путевок. Объединение выполняется по всем таблицам. Запрос ограничен параметрами, которые пользователь, являющийся HTTP-клиентом, указал в HTTP-строке запроса. Если поле не упоминалось в HTTP-запросе, оно не включается в SQL-запрос. Любая величина, переданная в HTTP-запросе, вносится в SQL-запрос. Для примера рассмотрим следующую HTTP-строку запроса:

```
http://server/travelds.asp?Country=Mexico&Month=May&Price=900
```

Она превращается в такой SQL-запрос:

```
SELECT * FROM Price tPrice, Packages tPkg, Availability tAvail
WHERE tPkg.PackagedId = tPrice.PackageId AND tPrice.PriceID = tAvail.PriceID AND
Month(tAvail.StarDate ) = 'May' AND tPkg.Country = 'Mexico' AND tPrice.Price <= 900
ORDER BY tPkg.Country, tPkg.City, tPkg.ResortName
```

Обратите внимание, что мы немножко смошенничали с месяцем. Дата заезда содержится в базе данных в виде полной даты, а запрашивается месяц. Чтобы извлечь значение месяца из базы данных, была использована функция `Month()`, которая может и не работать с базами данных, отличающимися от Microsoft Access.

Вдобавок для ясности мы не производим проверку вводимых данных на правильность. Это действительно проблема, потому что в формальные переменные можно превратить любой текст, и этот текст потом был бы вставлен в оператор `'where'` SQL-инструкции. Важно понять, что можно, разумеется, придумать решение, которое справится с этой проблемой, но, поскольку разработка будет доступна любому пользователю в Internet, следует подумать и о последствиях такого решения, на случай если кто-нибудь вдруг решит использовать ваше приложение для нужд, для которых оно не предназначалось.

После того как SQL-запрос сформирован, он выполняется в базе данных при помощи ADO. В итоге возвращается группа записей, содержащая результат. Производится цикл по записям и строится XML-документ, в котором информация из записи вставляется в соответствующие места.

Поскольку XML-элементы `country`, `city` и `resort` являются элементами-контейнерами, они обрабатываются особым образом. Необходимо расположить их в возвращаемых записях по порядку и создать открывающие и закрывающие тэги.

Одна из сложностей заключается в том, что в возвращенных записях некоторая информация может повторяться, особенно поля `country` и `city`. В XML-документе желательно указать такую информацию только один раз. Поэтому мы устанавливаем флаг, который сообщает, когда мы переходим к новому городу или стране, и только после этого в поток XML вставляется открывающий тэг для этих данных. Другая сложность – поставить закрывающий тэг на правильное место. Для этого снова можно использовать флаг. Фактически закрывающие тэги вставляются перед открывающими тэгами следующей записи для всех записей кроме первой. Еще одно условие – если имелись возвращенные записи, закрывающие тэги добавляются в конце документа.

Метеорологический сервер

Мы запускаем метеорологический сервер, посылая HTTP-запрос `GET`. Web-сервер запрашивает базу данных, в соответствии с запросом, и возвращает XML-документ с данными о погоде.

Ввод

Форма запроса такова:

```
http://server/weatherds.asp?Country=VAL1&City=VAL2&Month=VAL3
```

Таблица 11.2. Параметры запроса на метеорологический сервер

server	Имя домена метеорологического сервера
Country (optional)	Страна, для которой запрашивается погода (необязательно)
City (optional)	Город, для которого запрашивается погода (необязательно)
Month (optional)	Месяц, для которого запрашивается погода (необязательно)
VAL (1-3)	Величины для ограничения подбора совпадений

Вывод

Для вывода создается XML-документ, который придерживается нашего DTD для отчета о погоде. Он содержит результаты, соответствующие деталям переданного запроса. Если никакие параметры указаны не были, в документе возвращаются все записи. Рассмотрим пример:

```
http://server/ weatherds.asp?Country=Australia.
```

В этом запросе возвращается отчет о погоде во всех городах Австралии для всех месяцев года.

Реализация

Ниже дан код ASP для файла `weatherds.asp`. Он осуществляет всю необходимую работу от получения запроса до отправки назад XML-документа с результатами.

```
<%@ LANGUAGE=VBSCRIPT%>
<% Response.ContentType = "text/plain" %>
<%

' Строим SQL-запрос из переменных формы, переданной в URL.
Dim curCountry
Dim firstCountry
Dim curCity
Dim firstCity
Dim doEndTags

firstCountry = True
firstCity = True
doEndTags = False

SQLStatement = "SELECT * FROM weather tW, location tLoc WHERE tW.LocationId
                    = tLoc.LocationId"
If Request.QueryString( "Country" ) <> "Any" Then
   SQLStatement = SQLStatement + " AND tLoc.Country = '" + Request.QueryString(
```

```
"Country" ) + "'"
End If
If Len (Request.QueryString( "City" ) ) > 0 Then
   SQLStatement = SQLStatement + " AND tLoc.City = '" + Request.QueryString(
"City" ) + "'"
End If
If Len (Request.QueryString( "Month" ) ) > 0 Then
   SQLStatement = SQLStatement + " AND tW.Month = " + Request.QueryString(
"Month" )
End If
SQLStatement = SQLStatement + "ORDER BY tLoc.Country, tLoc.City"
%>

<?xml version="1.0"?>
<!DOCTYPE weather SYSTEM "weather.dtd">
<weather>
<%
   Set Conn = Server.CreateObject("ADODB.Connection")
   Conn.Open "weatherDB"
   set weatherRec = Conn.Execute(SQLStatement)

   ' Проходим циклом по записям, создающим элементы XML.
   Do While Not weatherRec.EOF

       doEndTags = True

       ' Начинаем новый элемент COUNTRY, если находимся в новой стране.
       If curCountry <> weatherRec("COUNTRY") Then
           If firstCountry = False Then
               Response.Write(vbTab + "</country>" + vbNewLine)
           End If
           curCountry = weatherRec("COUNTRY")
           Response.Write(vbTab + "<country name=" + Chr(34) + curCountry +
       Chr(34)+ ">" + vbNewLine)
           firstCountry = False
       End If

       ' Начинаем новый элемент CITY, если находимся в новом городе.
       If curCity <> weatherRec("CITY") Then
           If firstCity = False Then
               Response.Write(vbTab + vbTab + "</city>" + vbNewLine)
           End If
           curCity = weatherRec("CITY")
           Response.Write(vbTab + vbTab + "<city name=" + Chr(34) + curCity +
       Chr(34)+ ">" + vbNewLine)
           firstCity = False
       End If

%>
           <month name="<%=weatherRec("MONTH")%>">
               <precipitation cm="<%weatherRec("AVGPRECIPITATION")%>"/>
               <winds speed="<%=weatherrec("WINDS")%>"/>
               <high c="<%=weatherRec("HIGH")%>"/>
```

```
                <low c="<%=weatherRec("LOW")%>"/>
            </month>
<%
    weatherRec.MoveNext
    Loop

    ' Добавляем замыкающие тэги только если действительно имеются элементы.
    If doEndTags = True Then
        Response.Write(vbTab + vbTab + "</city>" + vbNewLine)
        Response.Write(vbTab + "</country>" + vbNewLine)
    End If
%>
</weather
```

Данный подход аналогичен подходу при создании XML-документа путевок. Одна из первых операций – это подготовка SQL-инструкции, которая затем должна будет выполняться. Поскольку элементы `country` и `city` являются внешними элементами в XML-документе, в SQL-инструкции мы задаем сортировку по ним, так что записи по одной и той же стране и городу будут сгруппированы. Для исполнения SQL-инструкции и возврата группы записей мы используем ADO. Затем проводим цикл по записям, извлекая требуемую информацию.

Как и при работе с XML-документом путевок, мы используем флаги, чтобы узнать, когда переходим к новой стране или городу и только после этого вставить в XML-поток открывающие тэги для страны или города. Закрывающие тэги, как и раньше, вставляются перед открывающими тэгами следующей записи для всех записей кроме первой. Если же имелись возвращенные записи, закрывающие тэги добавляются к концу документа.

Что делать дальше

Мы выполнили ту часть работы, которая касалась службы данных. Теперь пора переходить к выстраиванию системы бизнес-службы и службы клиентов. Бизнес-службы должны принимать запрос клиента, извлекать соответствующие XML-документы и преобразовывать их в один XML-документ, который и будет являться ответом. Службы пользователя будут принимать этот ответ и форматировать его для представления пользователю. Этот процесс проводится на сервере при помощи XSL, поскольку CSS и XML на стороне клиента пока еще не работают.

Бизнес-службы

Бизнес-службы исполняют правила от имени пользователя; их цель – формирование информации из данных. Следует учесть, что в контексте бизнес-служб предполагается, что пользователь – это, скорее всего, другое приложение. Бизнес-правила фильтруют, ищут и объединяют данные и синтезируют информацию, которой интересуется пользователь. Бизнес-службы реализуют возможности XML по автоматическому поиску, фильтрации и объединению разнообразной информации.

Бизнес-службы «Туристического маклера» выполняют следующие операции:

- принимают запрос пользователя о путевках и погодных условиях;
- извлекают информацию о путевках из различных источников;
- фильтруют данные о них;
- извлекают данные о погоде;
- фильтруют данные о путевках в зависимости от полученных данных о погоде;
- создают XML-документ, содержащий результаты;
- возвращают созданный XML-документ пользователю.

Теперь подробнее рассмотрим все эти пункты. Сначала нырнем в омут головой и испытаем приложение, а затем посмотрим, как оно реализовано. Для этого воспользуемся несложным примером.

Пример

Запустите браузер и посетите страницу http://webdev.wrox.co.uk/books/1525. Щелкните по **business services emo** (демонстрация бизнес-служб). Заполните форму (рис. 11.10) и щелкните по **OK**.

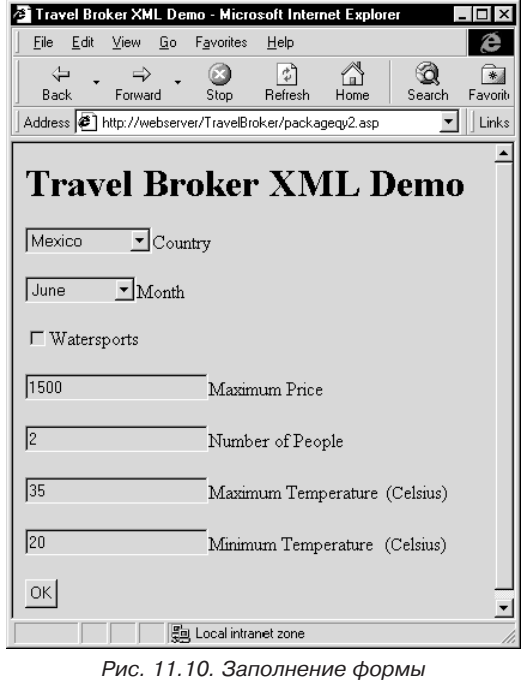


Рис. 11.10. Заполнение формы

В ответ вы получите необработанный XML-документ, приведенный ниже. Я добавил в него несколько переводов строки, чтобы облегчить чтение. Результаты, содержащиеся в документе, получены объединением информации от различных туристических серверов и отсечением неподходящих данных в соответствии с информацией о погоде.

```
<?xml version="1.0"?>
<travelpackages>
   <country name="Mexico">
      <city name="Cancun">
         <resort name="Crown Paradise Club Cancun" rating="4"
            typeofholiday="Beach"
            watersports="True" meals="True" drinks="True">
            <package dateofdep="6/28/98 price="1339"/>
            <package dateofdep="6/21/98 price="1319"/>
            <package dateofdep="6/14/98 price="1319"/>
            <package dateofdep="6/7/98 price="1319"/>
         </resort>
      </city>
      <city name="Puerto Vallarte">
         <resort name="Qualton Club and Spa" rating="4"
      typeofholiday="Beach"
            watersports="True" meals="True" drinks="False">
            <package dateofdep="6/26/98 price="1219"/>
            <package dateofdep="6/19/98 price="1089"/>
            <package dateofdep="6/12/98 price="1089"/>
            <package dateofdep="6/5/98 price="1089"/>
         </resort>
      </city>
   </country>
</travelpackages>
```

Этот документ очень похож на образец XML-документа, приведенный в разделе о службах данных, и действительно является состоятельным в соответствии с DTD путевок. Разница заключается в том, что данный документ был сформирован из множества XML-документов и затем сокращен в соответствии с некоторыми бизнес-правилами. Бизнес-правила пользуются данными из других XML-документов (полученных от метеорологического сервера) и данными, введенными пользователем.

Реализация

Инструменты, использованные для реализации, это Microsoft Internet Information Server 4 и Active Server Pages. В Active Server Pages был задействован VBScript. HTTP применялся в качестве протокола для перемещения XML-документов между бизнес-службами и их пользователем, так же как и между бизнес-службами и службами данных.

На схеме (рис. 11.11) изображен поток данных в приложении.

В нашем примере пользователь заполняет HTML-форму и щелкает по **OK**. При этом формируется HTTP-запрос, содержащий URL страницы ASP, которая реализует бизнес-службы, и формальные переменные. ASP создает экземпляр компонента COM `TBIntegrator`, который обладает всеми функциональными возможностями бизнес-служб. Формальные переменные присваиваются свойствам компонента COM, и далее реализуюся бизнес-правила. Затем из компонента COM возвращается XML-документ с результатами.

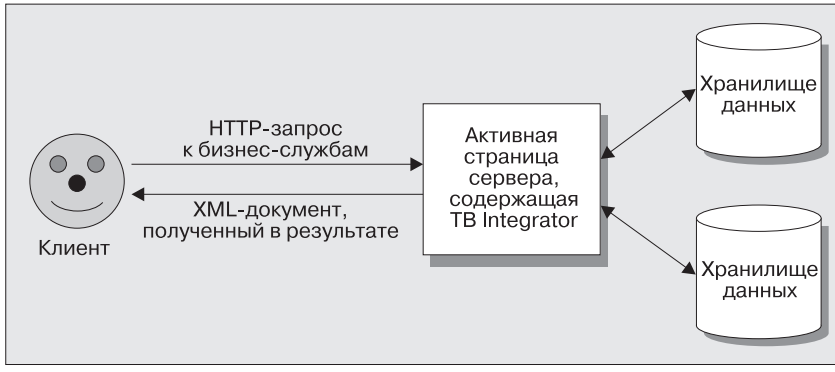


Рис. 11.11. Схема потока данных в приложении «Туристический маклер»

Компонент TBIntegrator

В нашем примере бизнес-службы реализованы компонентом COM, называемым `TBIntegrator`. Поскольку это COM-сервер, он может быть использован в любом COM-клиенте, будь то Visual Basic или Active Server Pages. Мы будем применять его в Active Server Pages. Клиент компонента `TBIntegrator`, являющийся для нас службой пользователя, устанавливает различные свойства компонента `TBIntegrator`. Эти свойства представляют собой запрос на сервер. По сути они являются ограничениями, которые выставляет конечный пользователь при поиске места для проведения отпуска. Затем вызывается метод для получения информации от различных служб данных, их фильтрации и объединения в XML-документ, который возвращается клиенту.

Проект для построения `TBIntegrator`, называемый `TBIntegrator.zip`, может быть получен с http://webdev.wrox.co.uk/books/1525. Это проект Visual C++ 6.0 вместе с ALT COM App Wizard. Загрузите файл из Сети, разархивируйте его в директорию, и вы сможете загрузить проект в Visual C++ 6.0. Загрузка и построение проекта должны быть возможны и в Visual C++ 5.0.

Когда в первой части главы мы занимались организацией службы данных, использовать для создания XML-документа можно было только стандартные компоненты. На самом деле компонент `TBIntegrator` извлекает и анализирует XML-документы, так что мы собираемся воспользоваться библиотекой Microsoft `msxsl.dll`, имеющей COM-компонент `XMLDocument`, с помощью которого можно осуществить разбор и управление XML-документом.

Библиотека MSXSL.DLL

XML-анализатор фирмы Microsoft входит в состав одной из динамически связываемых библиотек в браузере IE4, а именно, `MSXML.DLL`. К сожалению, эта разработка содержит все системные заголовки, что приводит к конфликту имен при импорте при помощи `#import`. Анализатор имеется также в компоненте XSL-обработки `MSXSL.DLL` производства Microsoft. Этот компонент импортируется без проблем.

Наш код использует анализатор, находящийся в компоненте Microsoft MSXSL.DLL. Самый простой способ установить этот компонент – запустить демонстрационную версию от Microsoft. Зайдите с помощью браузера Internet Explorer (Navigator даст ошибку) на http://www.microsoft.com/xml/xsl/XSLControlDemo/XSLControl.htm. Вас спросят, хотите ли вы установить этот компонент от Microsoft. Если вы ответите утвердительно, компонент управления MSXSL будет загружен и установлен на вашем компьютере.

Анализатор в компоненте MSXSL.DLL является неверифицирующим, в отличии от Java-анализатора Microsoft. Неверифицирующий анализатор проверяет, является ли документ правильным, но не проверяет его на состоятельность. Документ считается правильным, если он соответствует синтаксическим правилам XML. Документ считается состоятельным, если он еще и соответствует правилам своего DTD. Неверифицирующий анализатор работает быстрее, чем верифицирующий, потому что ему приходится производить гораздо меньше операций. Поскольку в этом случае используется неверифицирующий анализатор, мы полагаемся на составителя XML-документов, подразумевая, что тот проверит их на состоятельность. Бизнес-правила, применяемые в TBIntegrator, полагаются на документы, согласующиеся с DTD.

MSXSL.DLL используется как объект COM, доступ к нему осуществляется при помощи *библиотеки активных шаблонов* (Active Template Library, ATL). Использование следующей строчки в начале файла TBIntegrator.cpp реализует данный компонент:

```
#import <msxsl.dll>
```

Этой командой информация из библиотеки типов MSXSL включается в код, облегчая доступ к индивидуальным интерфейсам. Подробно компонент MSXSL в настоящей главе не описывается.

Заметим, что если необходимо включить информацию из msxml.h, вам потребуются приведенные ниже строки. Эти определения предотвращают конфликт между информацией, созданной из библиотеки типов, и файлом заголовков.

```
#define __IXMLElementCollection_INTERFACE_DEFINED__
#define __IXMLDocument_INTERFACE_DEFINED__
#define __IXMLElement_INTERFACE_DEFINED__

#include "msxml.h"
```

Подробности кода

Классы компонента TBIntegrator изображены на схеме (рис. 11.12).

Компонент TBIntegrator устанавливает информацию и затем вызывает метод FindPackages(). Сначала познакомимся с ITBIntegrator, единственным интерфейсом компонента. Этот интерфейс выставляет свойства и методы COM-контейнеру, которым в данном случае является Active Server Page. У интерфейса имеется несколько методов put для построения запроса. В нем отсутствует поддержка метода get, поскольку в нашем приложении нет необходимости возвращать информацию. Кроме того, компонент не модифицирует информацию,которая была передана при помощи put. Ниже (табл. 11.3) приведена устанавливаемая информация.

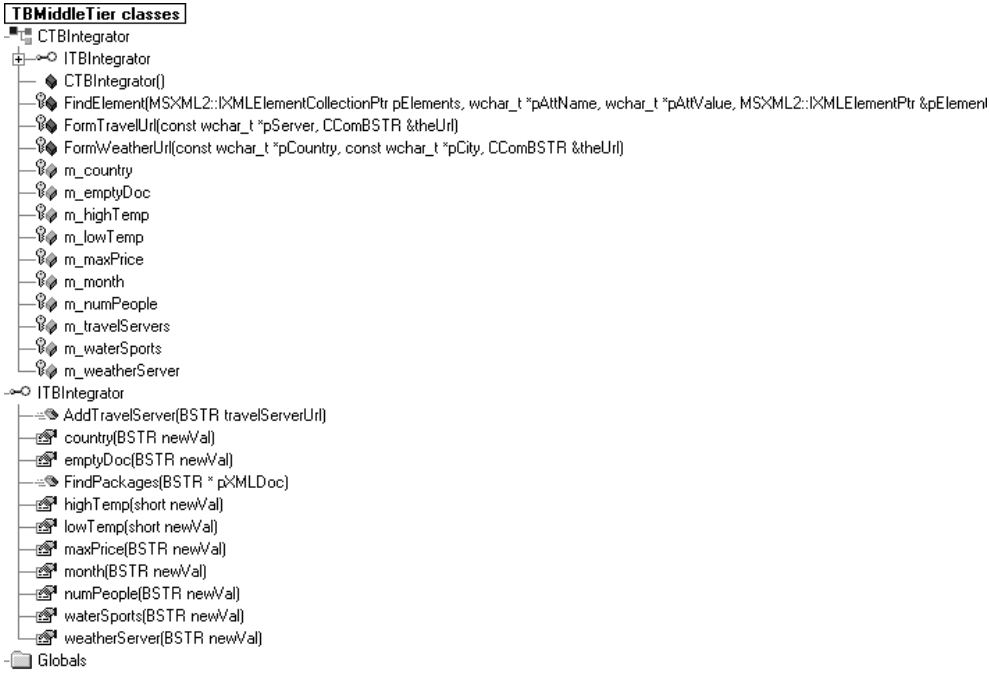


Рис. 11.12. Классы компонента TBIntegrator

Таблица 11.3. Установки интерфейса ITBIntegrator

Свойство	Описание	Обязательный
EmptyDoc	URL для XML-документа, который сервер использует в качестве шаблона для построения XML-документов	Да
Weather Server	URL Web-сервера, который предоставляет информацию о погоде в виде XML-документов. XML-метеорологический сервер был описан в предыдущей части главы	Да
Country	Страна, в которой следует искать курорты	Нет
MaxPrice	Максимальная цена путевки	Нет
Month	Месяц для поездки в отпуск	Да
NumPeople	Количество людей, собирающихся вместе в отпуск	Нет
WaterSports	Имеет или не имеет курорт водные виды спорта	Нет
Hightemp	Максимальная температура месяца	Да
Lowtemp	Минимальная температура месяца	Да

В дополнение метод `AddTravelServer` добавляет туристический сервер, который соответствует требованиям, предъявляемым к туристическому серверу; они описаны в предыдущем разделе этой главы. Компонент создает внутренний список таких серверов и затем получает информацию от каждого из них. Каждый туристический сервер представляет турбюро, непосредственно продающее путевки.

Всю работу проделывает метод `FindPackages`. Он использует все свойства, которые сконфигурировал компонент `ITBIntegrator` и создает XML-документ в соответствии со встроенными в нем правилами. Мы познакомимся с этими правилами, когда начнем изучать код в деталях.

Схема на рис. 11.13 показывает взаимоотношения между Active Server Pages, COM-компонентом `TBIntegrator` и библиотекой `MSXSL.DLL`. ASP создает экземпляр компонента `TBIntegrator` и устанавливает его различные свойства. Когда в компонент `TBIntegrator` поступает команда `FindPackages()` (НайтиПутевки), он использует `MSXSL.DLL`, чтобы создать XML-документ при помощи IXMLDocument2. Компонент осуществлет доступ к различным частям документа, используя интерфейсы `IXMLElementCollection` и `IXMLElement`.

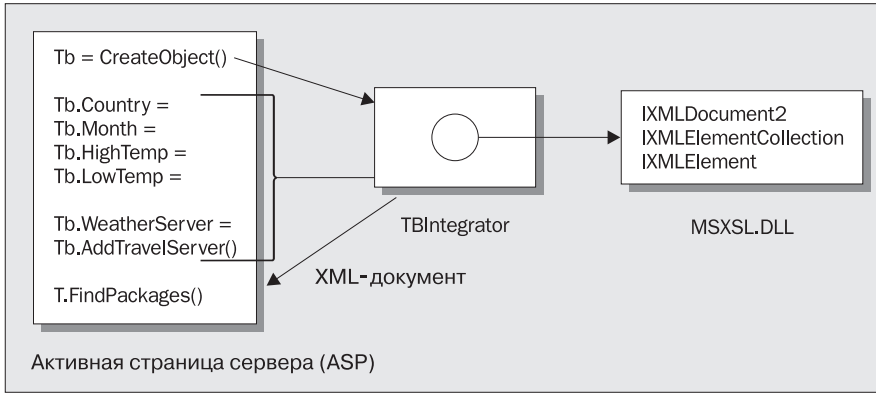


Рис. 11.13. Схема взаимодействия компонентов приложения «Туристический маклер»

С точки зрения языка C++, все свойства реализованы с переменными-членами. Когда вызывается метод `put`, переменной-члену присваивается новое значение. Туристические серверы представлены списком *библиотеки стандартных шаблонов* (Standard Template Library, STL). Когда вызывается `AddTravelServer()`, переданное значение добавляется в список.

В этом коде используются три защищенные функции-члена C++.

```
void CTBIntegrator::FormWeatherUrl(const wchar_t *pCountry, const wchar_t
*pCity, CComBSTR &theUrl) const
```

Этот метод создает URL для осуществления запроса XML-документа с метеорологического сервера. Он принимает базовый URL, установленный на свойство метеорологического сервера, и добавляет `country`, `city` и `month` в качестве параметров запроса.

```
void CTBIntegrator::FormTravelUrl(const wchar_t *pServer, CComBSTR &theUrl) const
```

Этот метод создает URL для выполнения запроса XML-документа с туристического сервера. Он принимает переданный базовый URL и добавляет в качестве параметров запроса свойства компонента: `country`, `month`, `watersports`, `maxprice` и `NumPeople`, если они были сообщены.

```
bool CTBIntegrator::FindElement( MSXML2::IXMLElementCollectionPtr pElements,
wchar_t
    *pAttName, wchar_t *pAttValue, MSXML2::IXMLElementPtr &pElement ) const
```

Этот метод является вспомогательным для нахождения элемента, основанного на значении атрибута. Набор XML-элементов передается вместе с именем атрибута и значением атрибута. Первый XML-элемент из набора, имеющий переданное значение атрибута, присваивается `IXMLElementPtr`. Если в наборе найден элемент с таким значением атрибута, возвращается `True`, если не найден, возвращается `False`. Можно надеяться, что подобные вспомогательные методы будут обеспечены в будущих реализациях XML-анализатора.

Итак, мы переходим к методу `FindPackages()`, который реализует логику компонента `TBIntegrator`. Для этого необходимо выполнить следующие операции:

- получить документ-шаблон, который будет возвращаемым документом;
- для каждого туристического сервера, установленного при помощи метода `AddTravelServer`:
  - сформировать URL запроса туристического сервера:
  - получить XML-документ, являющийся ответом на запрос;
  - слить XML-документ с возвращаемым документом;
- для каждой страны и города в возвращаемом XML-документе:
  - сформировать URL запроса метеорологического сервера;
  - получить XML-документ, являющийся ответом на запрос;
  - сократить возвращаемый XML-документ на основании данных о погоде;
- вернуть XML-документ.

Мы будем изучать код сразу по несколько строчек, чтобы он не выглядел слишком сложным. Учтите, что в этом случае используются COM-указатели. Они сбрасывают исключения, когда возникает ошибка, поэтому вся аналитическая работа происходит в блоке `try`.

```
STDMETHODIMP CTBIntegrator::FindPackages(BSTR * pXMLOut)
{
    HRESULT answer = S_OK;
    USES_CONVERSION;
    try
    {
        MSXML2::IXMLDocument2Ptr pReturnedDoc(__uuidof (MSXML2::XMLDocument2));
        MSXML2::IXMLDocument2Ptr pWeatherDoc(__uuidof (MSXML2::XMLDocument2));
        MSXML2::IXMLDocument2Ptr pTravelDoc(__uuidof (MSXML2::XMLDocument2));
        MSXML2::IXMLElementCollectionPtr              pTravelCountries(NULL);
        MSXML2::IXMLElementCollectionPtr              pReturnCountries(NULL);
        MSXML2::IXMLElementPtr                pReturnedRoot(NULL);
        MSXML2::IXMLElementPtr                pTravelRoot(NULL);
        MSXML2::IXMLElementPtr                pTCountryElement(NULL);
        MSXML2::IXMLElementPtr                pRCountryElement(NULL);

    CComBSTR                  theUrl;
    const CComVariant         countryTagName("COUNTRY");
```

```
const CComVariant          cityTagName("CITY");
const CComVariant          resortTagName("RESORT");
VARIANT             varEmpty;

// Инициализируем некоторые переменные.
varEmpty.vt = VT_EMPTY;
```

Для начала объявлены несколько «разумных» указателей ATL. Существует три XML-документа, которые будут активны в одно и тоже время: возвращаемый документ, документ с информацией о погоде и текущий документ с информацией о путевках. Кроме того, нам нужны некоторые наборы XML-элементов. Мы удерживаем указатель на корневом элементе возвращаемого документа и корневом элементе текущего документа путевок. Имена тэгов для `country`, `city` и `name` инициализированы как `const` переменные типа `variant`. Кроме того, нам нужен пустой экземпляр типа данных `variant`, который также инициализируется.

```
// Инициализируем создаваемый документ.
pReturnedDoc->URL = m_emptyDoc.m_str;
pReturnedRoot = pReturnedDoc->root;
```

В качестве отправной точки при построении возвращаемого документа используем шаблон XML-документа, который имеет созданные элементы, но не имеет конкретных элементов данных. Он выглядит так:

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

</travelpackages>
```

Чтобы получить шаблон, для возвращаемого документа в качестве свойства устанавливается URL. Корневой элемент возвращаемого документа извлекается из полученного документа.

```
// Строим документ XML путем опроса всех серверов турагентств.
SERVER_LIST::const_iterator travelIter;

for (travelIter = m_travelServers.begin(); travelIter !=
    m_travelServers.end(); travelIter++)
{
    FormTravelUrl((*travelIter).c_str(), theUrl);

    // Выборка документа о путевках.
    pTravelDoc->URL = theUrl.m_str;

    pTravelRoot = pTravelDoc->root;
    pTravelCountries =
        pTravelRoot->Getchildren()->item(countryTagName, varEmpty);
```

На следующем этапе производится цикл по всем туристическим серверам и запрос у них информации о путевках. Сначала из списка туристических серверов извлекается базовый URL, затем формируется URL-запрос и извлекается XML-документ. Кроме того, присваивается корень текущего документа путевок и набор стран.

```
for (int i = 0; i < pTravelCountries->length; i++)
{
```

```
pTCountryElement = pTravelCountries->item(CComVariant(i), varEmpty);

// Проверяем, не просматривали ли мы эту страну.
if (FindElement(pReturnedCountries, L"Name",
    _bstr_t(pTCountryElement->getAttribute(L"Name")), pRCountryElement))
{
    MSXML2::IXMLElementCollectionPtr   pTravelCities;
    MSXML2::IXMLElementCollectionPtr   pReturnCities;
    MSXML2::IXMLElemenPtr                      pTCityElement;
    MSXML2::IXMLElementPtr                     pRCityElement;

        pReturnedCities = pRCountryElement->Getchildren()->item(
            cityTagName, varEmpty);
        pTravelCities = pTCountryElement->Getchildren()->item(
            cityTagName, varEmpty);
```

Для каждого элемента `country` из набора элементов `country` проверяем, содержится ли уже такая страна в возвращаемом документе. Это может случиться, если на предыдущем туристическом сервере имелись путевки в эту конкретную страну. В том случае, если страна уже есть в возвращаемом документе, необходимо проверить города.

```
for (int j = 0; j < pTravelCities->length; j++)
{
    pTCityElement = pTravelCities->item(CComVariant(j), varEmpty);
    // Проверяем, не просматривали ли мы уже этот город.
    if (FindElement(pReturnedCities, L"Name",
        _bstr_t(pTCityElement->getAttribute(L"Name")),
        pRCityElement))
    {
        MSXML2::IXMLElementCollectionPtr           pResorts;
        MSXML2::IXMLElementPtr                             pResortElement;

        pResorts = pTCityElement->Getchildren()->item(
                resortTagName, varEmpty);
        for (int k = 0; k < pResorts->length; k++)
        {
            pResortElement = pResorts->item(CComVariant(j),
                varEmpty);
            pRCityElement->addChild(pResortElement, -1, -1);
        }
    }
```

Для каждого элемента `city` из набора элементов `city` проверяем, есть ли уже такой город в возвращаемом документе. Это опять может случиться, если на предыдущем туристическом сервере имелись путевки в данный город. Если город уже есть в возвращаемом документе, добавляем каждый курорт в элемент города в возвращаемом документе.

```
    else
    {
        answer = pRCountryElement->addChild(pTCityElement, -1, -1);
    }
```

Если в возвращаемом документе города еще нет, добавляем элемент `city`, который включает все свои дочерние элементы в элемент `country` в возвращаемом документе.

```
else
{
    answer = pReturnedRoot->addChild(pTCountryElement, -1, -1);
}
```

Подобным же образом, если страна не была найдена в возвращаемом документе, страна в текущем документе путевок добавляется в возвращаемый документ.

После цикла по всем туристическим серверам в возвращаемом документе находятся сведения обо всех путевках, соответствующих запросу пользователя. Компонент `TBIntegrator` собрал эту информацию из различных источников, но информация о погоде еще не учтена. На следующем этапе возвращаемый XML-документ используется в качестве источника данных для выяснения, какую информацию требуется получить с метеорологического сервера.

```
pReturnedCountries =
    pReturnedRoot->Getchildren()->item(countryTagName, varEmpty);

if (pReturnedCountries != NULL)
{
    // Сокращаем документ XML по данным о погоде.
MSXML2::IXMLDocument2Ptr weatherDoc(__uuidof(MSXML2::XMLDocument2));
MSXML2::IXMLElementCollectionPtr          pReturnedCities( NULL );
MSXML2::IXMLElementPtr                    pRCityElement( NULL );
MSXML2::IXMLElementPtr                    pWcountryElement( NULL );
MSXML2::IXMLElementPtr                    pWCityElement( NULL );
MSXML2::IXMLElementPtr                    pWMonthElement( NULL );
MSXML2::IXMLElementPtr                    pWLowElement( NULL );
MSXML2::IXMLElementPtr                    pWHighElement( NULL );
MSXML2::IXMLElementPtr                    pWeatherRoot( NULL );
MSXML2::IXMLElementCollectionPtr          pLowCollection( NULL );
MSXML2::IXMLElementCollectionPtr          pHighCollection( NULL );
    CComVariant                   firstRec( 0 );
    CComVariant                   lowVal;
    const CComVariant             lowTagName( "LOW" );
    CComVariant                   highVal;
    const CComVariant             highTagName( "HIGH" );
```

В возвращаемом документе, как было сказано выше, перечислены страны, приемлемые только по одному критерию. Понятно, что некоторые из них в соответствии с информацией о погоде должны быть удалены. Снова устанавливаются переменные и поиск продолжается. При присвоении имен переменным вставленная в имя буква `W` будет обозначать какую-то информацию в документе со сведениями о погоде, а буква `R` – информацию в возвращаемом документе. Например, `pWCityElement` является указателем на элемент города в XML-документе со сведениями о погоде.

```
for ( int = 0; i < pReturnedCountries->length; i++ )
{
    pRCountryElement = pReturnedCountries->item( CComVariant( i ),
        varEmpty );
    pReturnedCities = pRCountryElement->Getchildren()->item( cityTagName,
        varEmpty );

    for ( int j = 0; j < pReturnedCities->length; j++ )
    {

pRCityElement = pReturnedCities->item(CComVariant(j), varEmpty);

FormWeatherUrl(_bstr_t(pRCountryElement->getAttribute(L"Name")),
    _bstr_t(pRCityElement->getAttribute(L"Name")), theUrl);
```

Чтобы сократить возвращаемый документ, необходимо извлечь информацию о погоде в каждом городе.

```
weatherDoc->URL = theUrl.m_str;
pWeatherRoot = weatherDoc->root;

pWCountryElement = pWeatherRoot->Getchildren()->item(firstRec,
    varEmpty);
pWMonthElement = pWCityElement->Getchildren()->item(firstRec,
    varEmpty);
pLowCollection = pWMonthElement->Getchildren()->item(lowTagName,
    varEmpty);
pWLowElement = pLowCollection->item(firstRec, varEmpty);

lowVal = pWLowElement->getAttribute(L"C");
lowVal.ChangeType(VT_I2);

pHighCollection = pWMonthElement->Getchildren()->item(highTagName,
        varEmpty);
pWHighElement = pHighCollection->item(firstRec, varEmpty);

highVal = pWHighElement->getAttribute(L"C");
highVal.ChangeType(VT_I2);
```

Теперь можно считать, что документ со сведениями о погоде получен. Информация, необходимая для отсечения ветвей дерева документа путевок, собирается при обходе дерева документа. Конкретно, мы спускаемся до элемента `month`. Подробные данные о погоде являются дочерними элементами элемента `month`. Таким образом, мы можем без риска запросить первый элемент набора элементов `LOW`, которые являются дочерними к элементу `month`. Имея элемент `LOW`, мы можем получить действительную нижнюю предельную температуру в градусах Цельсия, запрашивая атрибут `C`. Температура возвращается в виде строки типа variant, и мы меняем ее тип на `integer`, чтобы в будущем облегчить операцию сравнения.

```
if (m_lowTemp > lowVal.iVal) || (m_highTemp < highVal.iVal))
{
        answer = pRCountryElement->removeChild(pRCityElement);
}
```

Если погодные условия неудовлетворительны, соответствующий элемент `city` удаляется из документа. Обратите внимание: это означает, что все дочерние элементы данного элемента `city` тоже удаляются из документа.

```
// У элемента COUNTRY нет потомков слева, удалить его.
if (pRCountryElement->Getchildren()->length == 0)
{
   pReturnedRoot->removeChild(pRCountryElement);
```

Очевидно, что при удалении из документа всех городов, в которых погодные условия неудовлетворительны, у элемента `country` не остается дочерних элементов. В этом случае удаляется и элемент `country`.

```
// Получить строку представления документа.
CComQIPtr<IPersistStreamInit, &IID_IPersistStreamInit> pPSI;
Istream     *pStream = NULL;
char            *pStr;
HGLOBAL     hGlobal = NULL;

answer = pReturnedDoc->QueryInterface(IID_IPersistStreamInit, (void **)&pPSI);

answer = CreateStreamOnHGlobal( NULL, TRUE, &pStream );
answer = pPSI->Save( pStream, TRUE );
answer = GetHGlobalFromStream( pStream, &hGlobal );
pStr = (char *)GlobalLock( hGlobal );
DWORD dwSize = GlobalSize( hGlobal );

CComBSTR    tempBStr( dwSize, pStr );

*pXMLOut = tempBStr.Detach();

GlobalUnlock( hGlobal );
```

Теперь XML-документ полностью обработан. Необходимо вернуть его вызывающей программе в виде строки, а не в виде бинарной структуры, в котором он находится в настоящий момент. `XMLDocument` поддерживает `IpersistStreamInit`, и, воспользовавшись этим интерфейсом, мы получим представление XML-документа в виде строки.

```
catch ( const _com_error &comErr )
{
   ATLTRACE( "Error:0x%x - %s\n", comErr.Error(), comErr.Description() );
   answer = comErr.Error();
}
```

И наконец, у нас есть *блок-ловушка* (catch), который будет принимать любые ошибки, возникающие во время обработки. Следует учесть, что на самом деле данная демонстрационная программа вообще не занимается погрешностями. В коммерческой системе потребовалась бы более серьезная обработка ошибок и восстановление приводимых данных после их исключения.

Теперь `TBIntegrator` завершен и готов к использованию. Впоследствии в этой главе мы вставим его компонент в страницу Active Server Page, которая свяжет бизнес-службы и службы пользователя.

Службы пользователя

Данные службы готовят информацию для представления конечному пользователю. С этой целью службы пользователя применяют правила форматирования. Вот два примера различных способов выводов информации: человеку с плохим зрением требуется более крупный шрифт, а тому, кто обращается за информацией по телефону, необходимо, соответственно, звуковое представление. Источник информации в обоих случаях один и тот же.

Способность XML разделять информацию и способ ее представления в значительной степени облегчают работы служб пользователя, тем более, что источник информации доступен в виде XML-документа. Службы пользователя определяют потребности каждого человека, обратившегося с запросом, и выбирают оответствующий механизм вывода. Представление XML-документа может осуществляться через CSS или XSL.

Пример

Еще раз обратимся к уже приводимому демонстрационному примеру (см. подраздел «Реализация при помощи ASP и ADO), а потом исследуем код, рализующий эту демонстрацию. Посетите страницу http://webdev.wrox.co.uk/books/1525. Щелкните на **user service demo**.

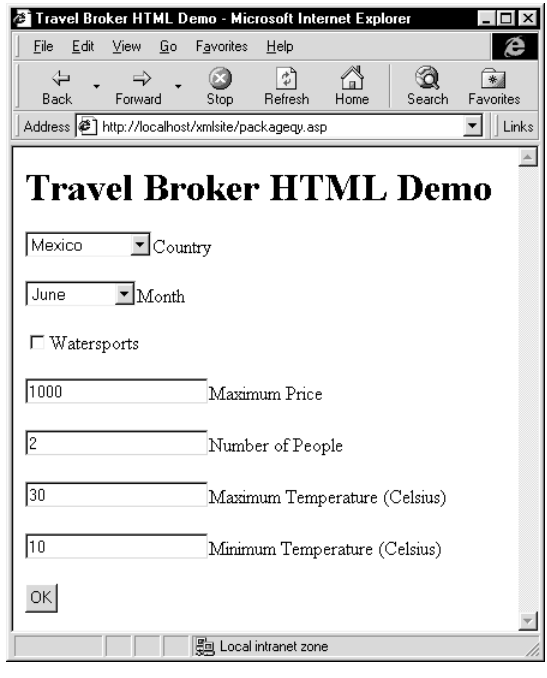


Рис. 11.14. Форма ввода данных служб пользователя

Как видите, форма (рис. 11.14) точно такая же, как в службах данных. Отличие состоит в том, что вместо XML-документа будет возвращен HTL-документ. Заполните эту форму и щелкните по **OK**. В ответ вы получите результаты в HTML-формате (рис. 11.15).

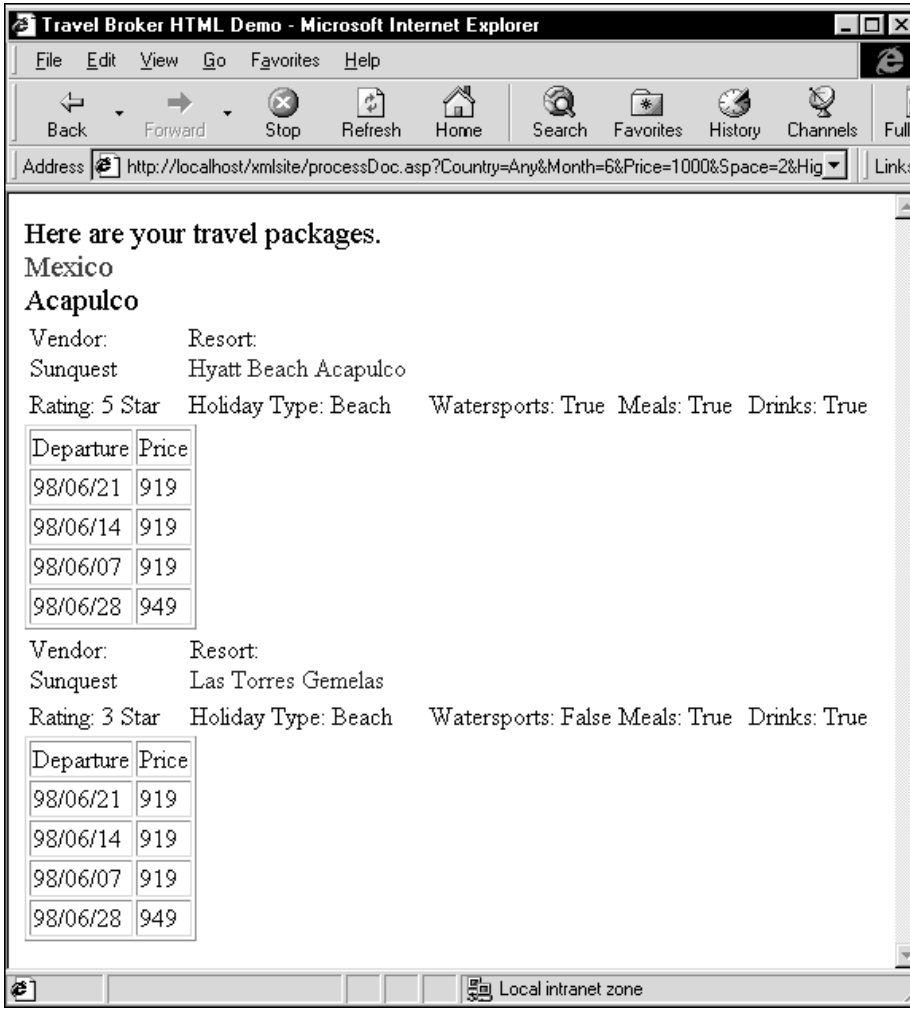


Рис. 11.15. Вывод запрошенной информации в службах пользователя

Форматирование у клиента при помощи CSS

Форматирование XML-документа на стороне клиента дает выгоду в смысле переноса части нагрузки с сервера на компьютер клиента. Это разделение обязанностей приводит к повышению эффективности всей системы в целом. В настоящее время IE5 является единственным браузером, предлагающим хотя бы какую-то поддержку этой опции. Поскольку на момент подготовки сборника этот браузер был доступен только в beta-версии, пример подобного форматирования не включен в данную книгу.

Форматирование на сервере при помощи XSL

Эта методика позволяет устранить влияние различий между браузерами. Сервер решает, какой в точности HTML-документ будет создан и послан браузеру

для представления. Однако поскольку Web-сервер при этом выполняет больший объем работы, эффективность системы в целом снижается.

Компонент `XSLControl` фирмы Microsoft позволяет форматировать XML-документ при помощи XSL. В результате получается HTML-документ. Для нашей реализации мы собираемся создать экземпляр `XSLControl` в Active Server Page на сервере. Он примет у бизнес-служб XML-документ с результатами поиска, применит к нему XSL-документ для форматирования и вернет полученный XML-документ в Active Server Page. Затем Active Server Page возвратит HTML-документ клиенту.

Метод создания HTML на Web-сервере подобен употребленному в следующей главе («Сорняки Эль Лимона»). Там, как и в нашем случае, браузер не имеет ни малейшего представления, как был сформирован конечный результат. Он просто получает HTML-документ. Два примера различаются тем, что «Туристический маклер» создает документ динамически, когда задается запрос, в то время как в приложении «Сорняки Эль Лимона» законченный HTML-документ создается заранее. Преимущество динамического создания документа состоит в том, что возвращаемые данные по новизне не уступают содержимому базы данных, а ведь именно этого мы и хотели добиться для системы запросов и бронирования путевок. Недостатком является большой объем обработки, который необходимо выполнить перед окончательным возвратом HTML-документа.

Если вы выполнили установку `msxsl.dll`, описанную в разделе о бизнес-службах, у вас уже есть установленный `XSLControl`.

Управляющий XSL-документ

XSL-документ управляет представлением документа. Рассмотрим каждое из правил.

```
<xsl>
   <rule>
       <root/>
       <HTML>
           <TITLE>Travel Package Results</TITLE>
           <BODY color="black" background-color="white" font-size="14pt">
               <H2>Here are your travel packages.</H2><BR/>
               <children/>
           </BODY>
       </HTML>
   </rule>
...
```

Правило `root` устанавливает контекст всего документа. Оно идентифицирует корень дерева документа-источника, устанавливая тэги, которые должны обрамлять документ.

```
...
   <rule>
       <target-element type="country"/>
       <DIV color="red">
           <eval>this.getAttribute("name")</eval>
       </DIV>
       <children/>
```

```
    </rule>
...
```

Это правило находит элемент `country` и устанавливает, что значение атрибута `name` будет выведено красным цветом.

```
...
    <rule>
        <target-element type="city"/>
        <DIV color="blue">
            <eval>this.getAttribute("name")</eval>
        </DIV>
        <children/>
    </rule>
...
```

Это правило находит элемент `city` и устанавливает, что значение атрибута `name` будет выведено синим цветом.

```
...
<rule>
    <target-element type="resort"/>
        <TABLE>
            <TR>
                <TD WIDTH="120">Vendor:
                <eval>this.getAttribute("vendor")</eval></TD
                <TD WIDTH="200">Resort:
                    <DIV color="purple">
                        <eval>this.getAttribute("name")</eval>
                    </DIV>
                </TD>
            </TR>
            <TR>
                    <TD WIDTH="120">Rating: <eval>this.getAttribute("rating")</eval>
                            Star</TD>
                    <TD WIDTH="200">Holiday Type:
                                    <eval>this.getAttribute("typeofholiday")</
eval></TD>
                    <TD WIDTH="135">Watersports:
                                    <eval>this.getAttribute("watersports")</
eval></TD>
                    <TD WIDTH="100">Meals:
                        <eval>this.getAttribute("meals")</eval></TD>
                    <TD WIDTH="100">Drinks:
                        <eval>this.getAttribute("drinks")</eval></TD>
        </TR>
        </TABLE>
        <TABLE BORDER="1">
            <children/>
    </TABLE>
</rule>
...
```

Это правило находит элемент `resort` и выводит значения его атрибутов. Для более наглядного представления значения помещаются в таблицу. Дочерними элементами элемента `resort` (курорт) являются путевки, доступные на этом курорте. Мы хотим поместить каждую путевку в строку таблицы, поэтому тэги элемента `<TABLE>` обрамляют элемент `<children/>`.

```
...
<rule>
   <target-element type="package" position="first-of-type"/>
       <TR>
           <TD>Departure</TD><TD>Price</TD>
       </TR>
       <TR>
           <TD><eval>this.getAttribute("dateofdep")</eval></TD>
           <TD><eval>this.getAttribute("price")</eval></TD>
       </TR>
</rule>
...
```

Как было сказано выше, каждая путевка помещается в строку таблицы. Мы хотим, чтобы у таблицы был заголовок, поэтому первая строка таблицы должна быть обработана особым способом. Данное правило как раз и осуществляет такую обработку по образцу:

```
<target-element type="package" position="first-of-type"/>
```

Этот образец идентифицирует первый представляемый элемент `package` в XML-документе.

```
...
<rule>
   <target-element type="package"/>
       <TR>
           <TD><eval>this.getAttribute("dateofdep")</eval></TD>
           <TD><eval>this.getAttribute("price")</eval></TD>
       </TR>
</rule>

</xsl>
```

Последнее правило занимается представлением всех элементов `package`, кроме первого. Элементы `package` размещаются по строкам таблицы.

Реализация

Страница Active Server Page содержит компонент `TBIntegrator`, объединяющий документы и форматирующий компонент `XSLControl`. Кроме того, Active Server Pages координирует информацию, которая передается между этими компонентами. Страница принимает запрос в форме:

```
http://server/travelds.asp?Country=VAL1&
Month=VAL2&Watersports=VAL3&Price=VAL4&Space=VAL5&HighTemp=VAL6&LowTemp
=VAL7
```

Таблица 11.4. Параметры запроса, принимаемого Active Server Page

server	Имя домена туристического сервера
Country (необязательный)	Страна назначения
Month (обязательный)	Желаемый месяц для проведения отпуска
Watersports (необязательный)	Истинно, если в путевку входят водные виды спорта
Price (необязательный)	Максимальная цена путевки
Space (необязательный)	Число людей, отправляющихся вместе в отпуск
HighTemp	Максимальная температура во время отпуска
LowTemp	Минимальная температура во время отпуска
VAL(1-7)	Величины для ограничения подбора. Если необязательная величина не указана, этот атрибут не ограничивает результаты

Рассмотрим код самой страницы.

```
<SCRIPT LANGUAGE=VBScript RUNAT=server>

' Создаем объединенный документ.
Set tbInt = Server.CreateObject("WROX.TBIntegrator.1")

If Request.QueryString("Country") <> "Any" Then
   tbInt.Country = Request.QueryString("Country")
End If

tbInt.month = Request.QueryString( "Month" )
tbInt.waterSports = Request.QueryString( "WaterSports" )
tbInt.maxPrice = Request.QueryString( "Price" )
tbInt.numPeople = Request.QueryString( "Space" )

If Len(Request.QueryString( "HighTemp" ) ) <> 0 Then
   tbInt.highTemp = Request.QueryString( "HighTemp" )
End If

If Len(Request.QueryString( "LowTemp" ) ) <> 0 Then
   tbInt.lowTemp = Request.QueryString( "LowTemp" )
End If

tbInt.emptyDoc = "http://localhost/xmlsite/empty.xml"
tbInt.weatherServer = "http://sageconsultants.com/TravelBroker/weatherds.asp"
tbInt.AddTravelServer=("http://sageconsultants.com/TravelBroker
           /travelds.asp")
tbInt. AddTravelServer= ("http:// localhost/xmlsite/travelds.asp")

xmlDoc = tbInt.FindPackages()

Set tbInt = Nothing
...
```

Эта часть осуществляет объединение XML-документа. Создается объект `TBIntegrator`, его свойства устанавливаются из значений `QueryString`. Шаблон для конечного XML-документа, URL метеорологического сервера и список URL туристических серверов жестко закодированы в странице Active Server Page. Наконец, для исполнения бизнес-правил вызывается метод `FindPackages`. Как работает метод `FindPackages`, мы видели ранее в этой главе. Если обработка данным мето-

дом прошла успешно, результирующий XML-документ возвращается с перечнем всех путевок, соответствующих всем критериям, указанным пользователем. Затем мы избавляемся от объекта `TBIntegrator`, устанавливая его в состояние «ничто» (`nothing`). Скорейшее высвобождение ресурсов повышает эффективность приложения.

```
...
' Записываем полученный документ XML в файл.
Set fs = CreateObject( "Scripting.FileSystemObject" )
Set aFile = fs.CreateTextFile( "pkData.xml", True )
aFile.Write( xmlDoc )
aFile.Close

Set fs = Nothing
...
```

К сожалению, этот код содержит некоторые необходимые «заплаты» между компонентами `TBIntegrator` и `XSLControl`. Из компонента `TBIntegrator` XML-документ возвращается в виде строки. Другой компонент, `XSLControl`, принимает XML-документ только в виде URL. Поэтому приходится записывать XML-строку в файл и передавать имя этого файла в `XSLControl`. К счастью, будущие версии `XSLControl` будут принимать XML-документы и в виде строки.

```
...
' Форматируем документ XML.
Set xslProc = Server.CreateObject( "XSLControl.XSLControl.1" )

xslProc.documentURL = "pkData.xml"
xslProc.styleURL = "http://localhost/xmlsite/render1.xsl"
Response.Write( xslProc.htmlText )

Set xslProc = Nothing

</SCRIPT>
```

На последнем этапе происходит форматирование XML- в HTML-документ. Создается объект `XSLControl`. Устанавливается URL исходного XML-документа, а также URL со стилями форматирования. Результирующий HTML-документ доступен как свойство-строка компонента. Он передается клиенту, а мы избавляемся от объекта `XSLControl`, устанавливая его в состояние «ничто» (`nothing`).

Заключение

В этой главе было детально рассмотрена первая часть приложения «Туристический маклер», касающаяся службы данных. Мы увидели, как база данных может быть инкапсулирована при помощи XML-документа. Мы также узнали, как с помощью XML можно опубликовать информацию в Internet через HTTP. Службы данных реализуют цель XML – публикацию правильных (в смысле XML) данных.

При реализации служб данных мы познакомились с тем, как:

- сконструировать два вида определений DTD для XML;
- конвертировать HTTP-запрос в SQL-запрос;

- отобразить ответ из базы данных в виде XML-документа;
- создать XML-документ при помощи Active Server Pages.

Приложение «Туристический маклер» построено на основе трехуровневой архитектуры, включающей службы данных, бизнес-службы и службы пользователя. Службы могут быть размещены на физически разных вычислительных машинах, при этом для связи между уровнями используется XML. При помощи такого аппарата можно добиться, чтобы приложение публиковало структурированные данные, автоматизировало поиск, фильтрацию и интеграцию информации из различных источников. Кроме того, XML позволяет отделить сами данные от способа их представления.

Для демонстрации XML-технологии мы использовали различные разработки фирмы Microsof: VBScript в Active Server Pages, доступ к базе данных с помощью службы данных, управляемой ADO. Бизнес-службы были реализованы при помощи Microsoft C++, который использовался в компоненте COM, созданном при помощи ATL и C++. И наконец, службы пользователя были реализованы при помощи VBScript в Active Server Pages и при помощи XSLControl производства фирмы Microsoft.

Ссылки для получения дальнейшей информации

Файлы для этой главы: http://webdev.wrox.co.uk/books/1525

Дискуссия о CSS в консорциуме W3C: http://www.w3.org/Style/CSS/

Дискуссия об XSL в консорциуме W3C: http://www.w3.org/Style/XSL/

Дискуссия об элементах и атрибутах: http://www.sil.org/sgml/elementsandAttrs.html

Информация фирмы Microsoft об XSL: http://www.microsoft.com/xml/xsl/

Глава 12. «Сорняки Эль Лимона»: заказная издательская Web-система на основе XML

В этой главе мы расскажем о Web-проекте, выполнить который нам удалось с помощью простой издательской Web-системы – Java-приложения, преобразующего набор XML-документов и изображений в набор Web-страниц. В нашем случае ставилась задача описать флору, в основном травы и сорняки, которые произрастают в окрестностях одной из деревень Доминиканской республики. Сначала нам пришлось создать множество почти одинаковых Web-страниц – задача, сходная с построением онлайнового каталога. Если вам когда-нибудь придется заниматься подобной проблемой, для начала можно скопировать конструкцию нашей системы. Этот проект кажется мне исключительно интересным, поскольку на нем я в качестве Web-мастера прошел путь от овладения навыками разработки индивидуальных Web-страниц до создания сложных и правильно построенных Web-сайтов.

Как мы попали в этот переплет

В январе 1998 года мы с Оливией отправились в Доминиканскую Республику, в деревню Эль Лимон, планы развития которой должны были стать основой интересного проекта по обустройству всего региона. Предполагалось, что мы будем оказывать содействие в прокладке линии электропередачи от небольшой гидроэлектростанции до деревни, однако оборудование запаздывало, и нам пришлось поискать себе другое занятие.

Очень скоро мы пришли к выводу, что жителям деревни не хватает знаний в области сельского хозяйства; особенно им недоставало навыков в систематизации и ведении специальных записей, касавшихся борьбы с сорняками. Мы заинтересовались этой проблемой и в качестве пробного шара попытались описать флору деревни, сделав упор на мелкие сорные растения. К тому моменту крестьяне уже получили возможность работать с переносным компьютером и только ждали момента, когда в районе будет налажена сотовая связь. Через нее, посредством сотовых телефонов, они собирались получить доступ к Internet. Поразмыслив, мы пришли к выводу, что наш каталог флоры следует создать в виде набора Web-страниц. В этом случае для всех желающих его можно было бы распространять не только через Web в онлайновом режиме, но и посылать на дискетах тем жителям деревни, у которых еще нет доступа к Internet.

Почему не годились простые Web-страницы

Еще до отъезда из Эль Лимона я прикидывал, в какой форме лучше всего представить наш гербарий в Web, и уже тогда пришел к выводу, что писать HTML-страницы вручную не имеет смысла. Заполнить около тридцати Web-страниц не так уж трудно, но вот в один прекрасный день день вы решаете, что все заголовки должны быть синими, а не зелеными, или вдруг обнаруживается, что некая часть вашего HTML-кода приводит к зависанию Web-браузеров. В этом случае при всякой модификации проекта вас будет ждать сомнительное удовольствие вручную, не теряя согласованности, переписывать все тридцать страниц.

Как известно, каждый Web-сайт представляет собой сочетание стиля и содержания. Стиль – это выбор цвета, расположения, способа подачи материала, а содержанием в нашем случае являются описания и изображения трав. Чтобы приложение смотрелось достойно с профессиональной точки зрения, необходимо было сохранить стилистическую согласованность страниц. В небольших проектах ценой очень строгой дисциплины еще можно достичь требуемого единства, но по мере их разрастания, привлечения новых разработчиков выполнить это условие будет все труднее и труднее.

К тому же каждому понятно, что Web-страницы должны постоянно развиваться. В зависимости от откликов пользователей или в результате собственных размышлений о безупречном дизайне, их обычно по несколько раз перерабатывают, так что воспроизведение от руки целой кипы документов HTML – это самый неудобный способ создания Web-сайта, какой только можно придумать. При таком подходе изменить стиль очень трудно, и подобные попытки могут оказаться пустой тратой времени. Всякое изменение содержания, когда, например, возникает необходимость включить новые растения или исправить ошибки в ранее наработанных материалах, вынуждает переписывать каждую страницу, касающуюся данного растения, а также предметный указатель и все другие материалы, имеющие отношение к новой информации; и все это надо проделать, не внося дополнительных ошибок.

Web-разработчик тоже не должен стоять на месте. Сначала я создавал Web-страницы просто потому, что умел это делать, и даже не задумывался о поддержке и обслуживании своих питомцев. Но в конце концов стало ясно, что для перехода к более сложным Web-проектам мне просто необходимо научиться автоматизировать процесс подключения стиля к содержанию.

Почему я выбрал XML

Решив воспользоваться каким-либо из ныне существующих методов автоматического подключения стиля к содержанию, я первым делом должен был привести содержание – в нашем случае изображения и описания трав – к виду, удобному для работы подобной программы. На этом пути передо мной открывался широкий выбор возможностей.

Например, описание растений нетрудно было составить на языке, близком к препроцессору C, который допускал бы внедрение макросов и вставку материалов в соответствии с некоторыми условиями. Решение простое, хотя и ограниченное,

поскольку все, чем здесь можно заняться, это заполнять пустые места в шаблоне. В результате поменять порядок, в котором представляется информация, бывает трудно или даже нереально.

Другой вариант – создание файла с разделителями-запятыми, но и это решение меня не устраивало. Трудность в использовании такого подхода, как, впрочем, и других способов, применяющих простейшие форматы данных, заключается в том, что «Сорняки...» для него слишком сложны. Файл с запятыми-разделителями хорош для простых данных, например для каталога звезд и их положения на небе. В нашем случае, однако, различные виды трав требуют для воспроизведения разные типы информации. Некоторые травы вообще ботанически не определены или имеют несколько народных названий. Каждому виду трав должен быть посвящен большой блок текста, в котором хотелось бы свободно использовать запятые и другие знаки пунктуации, а также без ограничений размечать текст гиперссылками. Обладая особо выдающимися способностями, можно было бы реализовать и такой подход, но созданная конструкция оказалась бы настолько хрупкой, что в итоге пришлось бы тратить много времени и сил на поддержание ее работоспособности.

Третье решение заключалось в использовании базы данных. Однако подобный метод окупается только тогда, когда в базе содержатся сотни, а то и тысячи записей, и поиск требуется вести по сложным критериям. Если бы я делал Web-сайт с сотнями описаний трав, притом динамический, с возможностью поиска по сложным запросам, использование базы данных дало бы мне неоспоримые преимущества. Но установка базы данных ради тридцати двух видов трав, обучение ее программированию и работе с ней, явно не оправдывали затраченных усилий. Кроме того, конструкция моей программы, раздельно осуществляющей интерпретацию XML, представление данных и создание HTML-документа, при желании позволяла и в будущем использовать подобный метод. К тому же XML тоже может работать с базами данных. Допускается поместить XML-текст в поля базы данных или использовать XML в качестве формата как для экспорта, так и для импорта информации из базы данных.

Другие возможные решения, включая и XML, подразумевают хранение информации в структурированной форме, аналогичной структуре базы данных. В этом случае обрабатывающая программа будет гибко и независимо идентифицировать и обработать каждую часть описания трав. XML как раз и предоставил мне простую основу для создания структурированного языка описания каталога.

Структурированный язык позволяет программе понять, какое значение имеет та или иная часть документа. Как-то, когда я бросил взгляд на коробку с компакт-дисками, стоявшую рядом с компьютером, мне пришла в голову занятная аналогия. Если бы я делал Web-страницу о музыкальном альбоме, я мог бы отсканировать его обложку и преобразовать изображение в JPEG-файл, а потом вставить картинку в страницу, чтобы ею мог полюбоваться всякий человек. Исключая, конечно, утративших зрение, а в этом-то и все дело! Компьютерная программа мало что может поделать с изображением; она уж точно не способна определить по нему название группы или песни. С другой стороны, я мог бы, пользуясь HTML и не применяя изображений вместо шрифтов, создать Web-страницу, которая выглядела бы точь-в-точь как обложка альбома. Тогда для поиска страницы нетрудно было бы использовать поисковую машину, набрав, например, название песни, а Web-браузер

мог бы прочитать Web-страницу слепому. (В HTML 4 особое внимание уделено его доступности для людей с физическими недостатками.) Но даже в этом случае программа поиска воспринимала бы страницу как обширный текстовый массив. Она не смогла бы определить, какие слова представляют собой название альбома, а какие имя исполнителя или информацию о правах, а XML как раз допускает хорошо структурированную разметку и позволяет заключать в тэги любую часть документа, а также приспосабливать его к сложной обработке. Говоря точнее, допускается выбрать из различных источников необходимые куски информации, объединить их, а затем представить результат в определенном стиле.

Конечно, я мог бы и сам разработать систему структурной разметки, однако XML дает пользователю куда больше преимуществ. Поскольку в настоящее время существует большое количество XML-анализаторов, у меня нет необходимости заниматься подобной работой. К тому же в ближайшие годы множество людей и организаций начнут широко использовать XML, и в сети непременно появится достаточно бесплатных и коммерческих средств редактирования и обработки XML, которыми смогу пользоваться и я. И еще, если мне удастся убедить людей, что мое описание типа документа (DTD) представляет собой удобный формат для обмена описаниями растений, неплохо будет обмениваться данными в этом формате с другими пользователями.

Почему я выбрал статические Web-страницы

Определившись с методом (действительно, XML оказался наиболее удобным средством для хранения описаний растений), я столкнулся с проблемой выбора – в каком виде представлять информацию пользователю: либо с помощью апплета Java, либо страниц, динамически создаваемых на сервере. Можно также использовать статические Web-страницы.

Апплет, подобно приложению, допускает графическое взаимодействие, но я не мог вообразить, зачем мне понадобилась бы его гибкость? В таком случае мне пришлось бы изобретать новый пользовательский интерфейс, а пользователям пришлось бы изучать его, и скорее всего они не стали бы тратить на это время. Время загрузки тоже представляло бы проблему. Поскольку размер типичного XML-анализатора в формате JAR составляет около 100 Кбайт, пользователям, имеющим низкую скорость соединения (а таких большинство), пришлось бы, глядя на серую рамку, при особом везении ждать не менее минуты, пока загружается апплет. Большинство из них ретировались бы, не дожидаясь окончания процесса. Когда современные XML-анализаторы и обогащенные возможностями Java API будут включаться в состав Web-браузеров, тогда, наверное, и появится смысл писать небольшие апплеты, использующие XML. Сегодня подобный подход может помочь только в том случае, если пользователи заранее готовы примириться с задержкой во времени. (Да, в MSIE 4.0 действительно включен MSXML версии 1.0, но MSXML 1.0 не совместим ни с окончательной спецификацией XML, ни с современным MSXML версии 1.8.) Как только вы сочтете, что какая-то часть кода должна выполняться самим клиентом, вас ожидает еще одна проблема: придется решать, поддержкой каких именно Web-браузеров вы намерены ограничиться. Чем более тонкие особенности браузера вы используете, тем уже круг ваших клиентов.

Другое важное преимущество XML – это динамическое создание Web-страниц на сервере. Этот вариант дает возможность каждому пользователю работать со своим персональным представлением страниц, на которых воспроизводится каталог трав, или создать интерактивное приложение, которое позволяло бы всем, кто вошел на этот сайт, вносить свои комментарии или добавлять описания новых растений. Однако динамическое формирование Web-страниц оказалось бы выгодным только в том случае, если бы число описываемых трав было так велико (тысячи), что хранить и поддерживать на сервере столько статических страниц стало бы неудобно. Поскольку такая программа исполняется на сервере, клиенту нет необходимости использовать строго определенный тип браузера, и время на пересылку программы не расходуется. Хотя потенциал динамического подхода огромен, я все же решил отказаться от него, поскольку на уровне функциональных возможностей, необходимых для решения поставленной мной задачи, этот способ слишком усложнял работу.

В конце концов я решил остановиться на статических Web-страницах. Хотя статические Web-страницы и не самое изощренное средство, они просты и хорошо отработаны. Web-сервер, обслуживающий статические страницы, может принять куда больше обращений и имеет гораздо меньше причин для сбоев. Поскольку подобный механизм не требует специальных программ, расположенных на сервере, любой пользователь сможет создать зеркальный сайт; страницы при этом допускается записывать на гибкие диски или CD-ROM. И последнее – я остановил свой выбор на статических Web-страницах, поскольку хотел создать узел, который бы выглядел и действовал как печатный документ, а набор статических Web-страниц как раз обладает необходимым внешним видом и производит хорошее впечатление. Важно помнить, что это решение позволяет в любой момент перейти к другому представлению каталога. Имея данные в структурированной форме, я всегда могу написать новую программу, отображающую их в ином виде.

Почему я выбрал Java

Программу, конвертирующую XML-документы с описаниями растений в HTML, я решил писать на языке Java прежде всего потому, что хорошо знаком с ним и уже не раз пользовался этим инструментом. Java – простой и строго типизированный язык, с его помощью легко и приятно строить объектно-ориентированные конструкции. К тому же Java – дитя Internet, и в Web-проекте на его долю приходится много функций. Об апплетах Java слышали все, но не все понимают, что с помощью этого языка можно создавать и обычные приложения, вроде моей программы, а также сервлеты Java, которые используются на Web-серверах в качестве эффективной замены CGI-скриптов.

В конце концов я пришел к выводу, что если бы мне пришлось писать HTML-страницы от руки, без помощи соответствующих программ, то осуществить этот проект было бы чрезвычайно трудно. Поэтому мне необходимо было создать свою издательскую систему, автоматизирующую подключение стилей к содержанию. Я решил создать описания трав на XML и написать Java-программу для компиляции XML-описания в набор статических Web-страниц.

Создание XML-документа

Выбор XML-анализатора

Написать собственный XML-анализатор для меня не являлось проблемой, однако в случае с электронным гербарием мне хотелось воспользоваться уже существующим, так что для начала я задался главным на этой стадии вопросом: какой анализатор выбрать, верифицирующий или неверифицирующий. Верифицирующий анализатор сверяет разбираемый документ с определением типа документа (DTD) и отказывается анализировать несостоятельный документ. Его недостаток состоит в том, что он более громоздок и занимает значительный объем памяти; понятно, как важен этот критерий, когда пользователю необходимо загружать анализатор из Сети. К тому же верифицирующий анализатор медленнее работает. Однако поскольку я собирался писать свои XML-страницы вручную (и конечно, в них неизбежно появлялись бы ошибки), а при таком подходе скорость обработки и память большого значения не имеют, я все-таки решил прибегнуть к помощи верифицирующего анализатора. Если бы я занялся организацией апплета, тогда действительно имело бы смысл в процессе создания протестировать документы верифицирующим анализатором, но при этом в самом апплете уже использовать неверифицирующий анализатор, чтобы сберечь память, пропускную способность и время без опасений, что несостоятельный документ испортит работу. В начале февраля было доступно несколько бесплатных XML-анализаторов на языке Java, среди которых особо выделялся анализатор фирмы Microsoft, MSXML на языке Java версии 1.8. Прежде всего он был хорошо документирован. Поскольку рабочая спецификация XML изменилась после выпуска MSIE4.0, анализатор MSXML/Java версии 1.0, встроенный в MSIE4.0, оказался несовместимым с версией 1.8. (MSXML/Java можно загрузить с адреса http://www.microsoft.com/xml/parser/jparser.asp, где также находится дополнительная информация об этом анализаторе).

Ключевым фактором в решении выбрать анализатор MSXML сыграла доступность для него исходного кода. Хотя анализатор MSXML хорошо документирован, даже в самой выверенной документации случаются ошибки и пропуски, а в моем случае одного беглого взгляда на исходные коды было достаточно, чтобы получить окончательный ответ на большинство спорных моментов. Другими словами, все программы имеют дефекты, особенно программы, находящиеся в бета-версии или использующие авангардные технологии, однако, имея исходные коды, зачастую можно всего за несколько минут исправить несущественные ошибки. Когда, например, я обнаружил, что исходный образец MSXML не считывает внешние компоненты в определение DTD, что затрудняет импорт в документ списка иностранных символов с надстрочными знаками, я быстро просмотрел исходные коды, изменил одну строку кода и разрешил эту проблему.

Пример документа

Программистам на XML, которым приходится обрабатывать файлы, основанные на стандартных определениях DTD (например, MathML, CDF или RDF) или на существующих определениях DTD, созданных другими разработчиками,

не требуется лично заниматься подобным трудоемким процессом.

Я же поставил себе цель написать собственное определение DTD, исходя из представления о том, как, по-моему, должен выглядеть окончательный XML-файл. Мне было известно, что в работе понадобятся тэги для названия семейства, а также для латинского и общеупотребительного наименования каждого растения. Кроме того, надо было разметить описание растения тэгами для языка и источника. Эти предварительные наметки оказались близки к тому, каким оказался данный XML-код на сегодняшний день:

```
<?xml version="1.0"?>
<!DOCTYPE plantdata system "limon.dtd">
<plantdata>
   <species id="6">
<family>Cucurbitacea</family>
<latin>Momordica charantia L.</latin>
<common>balsam pear</common>
<common>balsam apple</common>
<common>cerasee bush</common>
<common xml:land="es>archucha</common>
<common xml:land="es>balsamina </common>
<common xml:land="es>achochilla</common>
<common xml:land="es>pepinillo</common>
<common xml:land="es>cunde amor</common>
<common xml:land="es>melao de Sao Caetano</common>
<common xml:land="es>carcilla</common>

<text type="description" source="Direnzo98">
Vine, climbs by tendrils. Leaves are alternate, soft and lighly hairy. Leaves are
deeply lobed with five lobes. (Length about <cm>3</cm>) Yellow flowers arise from
leaf axils as do tendrils. Flower has five petals, bright orange small clusters of
pistils and stamen at center. (Diameter about <cm>1.5</cm>) Pods are oval tapering
to a point with rows of little spikes, green turning orange as they mature.
Exploded pods show bright orange peels and four red seeds. Inside is sticky. Pod
length (about <cm>2.5</cm>) Stem is hairy, very hairy at terminal end. Found
growing on fence along main road in full sun.
</text>
</species>
</plantdata>
```

Определение типа документа

Определение типа документа (DTD) – это точное описание формата некоторого класса XML-документов. Поскольку я начинал с нуля, в первую очередь мне было необходимо заняться этим определением. У меня не было опыта работы с XML или MSXML, и потому я писал DTD по одному тэгу за один прием. Каждый раз, когда я добавлял туда новую запись, расширяющую возможности проекта, мне приходилось вносить несколько строк кода и в Java-класс MSXMLSpecies-Factory – в ту часть моей программы, которая была ответственна за понимание дерева разбора, выдаваемого MSXML. Это была прекрасная школа пошагового изучения XML, ведь у меня в руках была постоянно работающая программа, с

которой можно было экспериментировать. Когда прототип приложения был готов, я внес в него некоторые изменения, чтобы лучше согласовать мое определение DTD с XML и стандартами, связанными с ним. Теперь мое определение DTD выглядит так:

```
<!ELEMENT plantdata ( species )+>
<!ELEMENT species ( family?, latin*, common*, text*, cite*)>
<!ATTLIST species id CDATA #REQUIRED>

<!ELEMENT family ( #PCDATA )>
<!ELEMENT latin ( #PCDATA )>
<!ELEMENT common ( #PCDATA )>
<!ATTLIST common xml:lang CDATA "en">

<!ELEMENT text ( #PCDATA | a | cm | ref )*>
<!ATTLIST text type CDATA #REQUIRED>
<!ATTLIST text source CDATA #REQUIRED>
<!ATTLIST text xml:lang CDATA "en">

<!ELEMENT a (#PCDATA)>
<!ATTLIST a href CDATA #REQUIRED>
<!ATTLIST a xml:link CDATA #FIXED "simple">

<!ELEMENT cm (#PCDATA)>

<!ELEMENT ref EMPTY>
<!ATTLIST ref id CDATA #REQUIRED>

<!ELEMENT cite EMPTY >
<!ATTLIST cite source CDATA #REQUIRED>
<!ATTLIST cite page CDATA "">

<!ENTITY % ISOlat1 PUBLIC
   "ISO 8879-1986//ENTITIES Added Latin 1//EN//XML"
   "ISOlat1.pen">
%ISOlat1;
```

Главные решения

Спецификация XML дает возможность создавать огромный диапазон типов документов. Например, XML позволяет определению DTD содержать рекурсивные описания элементов. Это просто великолепно для XML-документов, представляющих древоподобную структуру, но я заранее решил, что постараюсь в любом случае избегать рекурсии, потому что подобный механизм мне не нужен и только добавит лишние проблемы. Я решил, что общая структура моего документа должна походить скорее на компонент базы данных. У меня будет несколько полей, в которые я могу заносить такие свойства, как семейство растений, латинское и народное названия.

Я решил, что будет правильнее наделить содержание тэга `<species>` (вид), в котором определяется растение, жесткой структурой. Сначала будет указываться обозначение семейства, затем латинское название, народное название, текстовые блоки с описанием растения, список с библиографическими ссылками – каждый раз в одном и том же порядке. Я решил, что содержание тэга `<text>` (текст) будет

менее структурированным. Оно очень похоже на HTML и представляет собой обычный текст вперемежку с небольшими наборами тэгов для введения ссылок на другие растения, гиперссылок на произвольные URL и для создания единиц числовых величин.

Описание элементов

Новые элементы задаются с помощью тэга `<!ELEMENT>`. Соотношения между тэгами определяются при помощи синтаксиса, аналогичного обычным выражениям, например, определение элемента

```
<!ELEMENT species ( family?,latin*,common*,text*,cite*)>
```

означает, что элемент `species` может содержать один или ноль `(?)` элементов `<family>` (семейство), ноль или больше `(*)` элементов `<latin>` (латинское название), `<common>` (народное название), `<text>`, `<cite>` (цитирование). Я также задал атрибут

```
<!ATTLIST species id CDATA #REQUIRED>
```

который сообщает XML-анализатору, что каждый элемент `<species>` обязан иметь атрибут `ID`, указанный, например как `<species id="6">`. Кроме того, необходимо было решить, какая информация должна содержаться в атрибутах, а какая в дочерних элементах. И дочерние элементы, и атрибуты могут быть описаны как обязательные, так что было бы разумно, например, сделать `ID` атрибутом или ввести новый элемент `<id>`, содержащий `ID` семейства. Я выбрал атрибут, поскольку это позволило мне немного сэкономить на количестве вводимых символов. Однако по сравнению с дочерними элементами, атрибуты имеют два существенных ограничения, которые заставили меня поместить всю оставшуюся информацию элемента `<species>` в дочерние элементы. Во-первых, атрибут может принимать только одно значение, а в элемент можно поместить переменное число дочерних элементов. У растений может быть более, чем одно латинское или народное название, поэтому я решил, что эти величины должны быть указаны как элементы. Наименование семейства у каждого растения может быть только одно, однако для общей согласованности я решил разместить эту характеристику в дочернем элементе вместе с латинскими и народными названиями. Вообще говоря, можно было бы несколько названий поместить в атрибут в виде одной строки, используя запятые как разделители, но что-то подсказало мне, что это будет неудобно и может привести к ошибкам.

Элементы `<family>`, `<latin>`, `<common>`, `<text>` обрамляют некий текст, наподобие `<family>Cucurbitacea</family>`, поэтому мы сообщаем XML-анализатору, что элемент `<family>` содержит обыкновенный текст, который XML называет `#PCDATA`. Мы определяем элементы как:

```
<!ELEMENT family ( #PCDATA )>
```

В справочниках мы нашли названия некоторых растений на английском, испанском и португальском языках. Чтобы указать язык для каждого народного названия, мы использовали стандартный атрибут `"xml:lang"`. Я решил, что разметка языковых различий тэгами должна основываться на стандартах. При этом можно

воспользоваться выгодами конструкции, тщательно разработанной экспертами, и одновременно быть уверенным, что обрабатывающие XML-код программы других разработчиков будут автоматически понимать мои тэги, касающиеся языка. Атрибут `"xml:lang"` встроен в XML. XML-анализатор, подчиняющийся стандартам, позволит любому тэгу в документе содержать атрибут `"xml:lang"`, даже если он явно не указан, но, отметив его явно, я выставил значение по умолчанию, улучшил совместимость с другими анализаторами, которые не имеют недавно введенного атрибута `"xml:lang""`, и уведомил всех, кому придется читать мое определение DTD, что собираюсь использовать это свойство. В ранних версия моего определения DTD я использовал для `<common>` и `<text>` атрибут `'LANG'`, который определил сам, следуя соглашениям для HTML 4.0 об обозначении языка. Впоследствии, узнав об `"xml:lang"`, я заменил прежний вариант.

Совет *Значения, которые может принимать атрибут xml:lang, определены в Internet Engineering Task Force's RFC1776 (RFC документы можно получить с http://www.ietf.org/), основанном в свою очередь на стандарте ISO 639. Список названий языков находится также по адресу http://www.sil.org/sgml/iso639a.html.*

Элемент `<text>` обладает более сложной структурой, чем другие дочерние элементы, потому что он, подобно элементу `<species>`, сам содержит дочерние элементы. Однако в отличие от элемента `<species>`, элемент `<text>` содержит дочерние элементы не в строго определенном порядке, а позволяет любое сочетание текста и дочерних элементов, очень напоминая этим HTML. Этот элемент определяется так:

```
<!ELEMENT text ( #PCDATA | a | cm | ref )*>
```

где `(#PCDATA|a|cm|ref)` означает `"#PCDATA` или любой из элементов `<a>`, `<cm>` или `<ref/>"`, а знак звездочки `(*)` показывает, что указанные элементы могут встречаться ноль или более раз. Таким образом, элемент `<text>` содержит смесь `#PCDATA` и этих трех элементов. Тэг `<a>` обеспечивает возможность гиперссылки, наподобие тэгу `<A>` в HTML, который разрешает ссылку на любой URL. Синтаксис тэга `<a>` соответствует предложению об XML-ссылках (см. http://www.w3.org/TR/1998/WD-xlink-19980303). Поскольку я использовал стандартный синтаксис для гиперссылок, другие XML-приложения, такие как поисковые роботы, будут способны обнаружить в моем документе ссылку и перейти по ней.

Тэг `<ref/>` был применен, чтобы снабдить внутренние ссылки из одного описания растения на другое ключами по ID-номеру растения, на которое указывает ссылка. В отличие от тэга `<a>`, я хотел, чтобы текст гиперссылки – либо первое латинское название растения, либо ID-номер, если растение не идентифицировано, – был фиксированным. Поэтому решил сделать тэг `<ref/>` пустым, чтобы он ничего не содержал, кроме атрибута `id`, указывающего, на какое растение осуществляется ссылка. Тэг `<cm>` помещен, чтобы отметить величины, измеряемые в сантиметрах, и писать `<cm>5</cm>` вместо «5 см». Это позволит программе обнаруживать значения числовых величин и конвертировать их, например, из сантиметров в дюймы.

Элемент <text> также содержит несколько атрибутов. Хотя мне пока и не довелось ими воспользоваться, я решил не упускать возможность работать более чем с одним текстовым блоком для каждого растения. Например, мы отыскали книги, которые описывают растения как на английском, так и на испанском языках, и хотели поместить на нашем Web-узле описания на испанском, если получим на это разрешение издателя. Кроме того, у нас также есть дневники наблюдений. Они тоже представляют определенный интерес, но мало сочетаются с описаниями, которые когда-либо могут быть добавлены на сайт. В будущем можно попросить кого-нибудь перевести эти описания на испанский язык. Если подобный дополнительный текст найдет свое место в нашей системе, хотелось бы иметь возможность сообщать в нашей программе, что вставлять в Web-страницы надо, например, только английский или только испанский текст, с тем чтобы создавать различные версии для разных языков, или включать в страницы только наши собственные описания растений. Чтобы все это оказалось возможным, мне пришлось добавить атрибуты `source` (источник) и `type` (тип) и сделать явным атрибут `"xml:lang"`, помечающий язык. Как и в случае с атрибутом `id` в элементе <species>, я мог бы расположить эту информацию в дочерних элементах элемента <text>. Однако скоро мне пришло в голову, что будет вернее отделить сам текст, помещаемый в элемент <text>, от информации о тексте (метаинформации), которая содержится в атрибутах элемента <text>.

Как и атрибуты элемента <text>, тэг <cite> также построен с расчетом на будущее. Моя текущая программа создания Web-сайта не обрабатывает этот тэг; по умолчанию механизм, обрабатывающий XML, игнорирует тэги, которых не понимает. Тэг <cite> указан в определении DTD, так что MSXML проверяет его состоятельность. MSXML также предоставляет объекты `Element` в дереве документа, которое оно передает моей программе, но, когда моя программа проходит по дереву документа, она никогда не запрашивает элементы <cite>, так что они игнорируются. Однако тэг <cite> может оказаться полезен тем, что предоставляет место для отслеживания библиографических ссылок внутри «Сорняков...». Даже сейчас я могу прочитать их, просматривая XML-файл в текстовом редакторе, а в будущем могу обновить свою программу, чтобы она автоматически добавляла гиперссылки на библиографию.

Две последние строки в DTD определении описывают параметрический компонент:

```
<!ENTITY % ISOlat1 PUBLIC
"ISO 8879-1986//ENTITIES Added Latin 1//EN //XML"
"ISOlat1.pen">
%ISOlat1;
```

чтобы импортировать список определений компонентов из внешнего файла, часть которого выглядит примерно так:

```
...
   <!ENTITY Agrave "&#192;">          <!--capital A, grave accent -->
   <!ENTITY Aacute "&#193;">          <!--capital A, acute accent -->
   <!ENTITY Acirc "&#194;">           <!--capital A, circumflex accent -->
   <!ENTITY Atilde "&#195;">          <!--capital A, tilde -->
...
```

Цель этого файла – импортировать определения символов с надстрочными значками, используемых в таких европейских языках, как испанский. Это стандартные определения компонента в HTML, и задача состоит в том, чтобы такие символы воспринимались в моем XML-документе.

Я получил файл `ISOlat1.pen` от Рика Джелиффа (Rick Jelliffe), который послал мне e-mail, увидев начальную версию моего определения DTD, включающую всего несколько определений символьных компонентов, которые я добавил вручную для двух символов с ударением, встретившихся мне в народных испанских названиях моих растений. Простым включением этого файла в мое определение DTD я сделал свой документ прозрачным для компонентов, используемых для символов с ударением в HTML 4.0; теперь мне легко вставлять иностранные тексты. `ISOlat1.pen` вместе с другими аналогичными файлами, такими как `ISOlat2.pen` и `ISOlat3.pen`, которые предоставляют доступ к другим частям набора компонентов HTML 4.0, можно загрузить с Web-сайтов Wrox/Honeylocust. (См. в конце главы раздел «Построение «Сорняков...»).

Небольшая проблема, которая мне встретилась во время описания DTD, заключалась в том, что MSXML версии 1.8 не поддерживает внешние параметрические компоненты внутри определения DTD. Поскольку фирма Microsoft любезно предоставила исходный код, я решил проблему, внеся изменения в одну строку кода и перекомпилировав его. Эти действия будут обсуждены позднее в разделе «Доработка исходного кода MSXML».

Эволюция

Поскольку я приступил к работе, не имея опыта работы с XML, мне пришлось, как уже было сказано выше, писать свое определение DTD строчка за строчкой. Начал я с одного XML-файла и некого чернового варианта Java-класса, являющегося интерфейсом между MSXML и оставшейся частью программы, MSXMLSpeciesFactory, и понемногу добавлял элементы в определение DTD, XML-файл и код. Это очень помогает погрузиться в проблему и быстро изучить XML, одновременно создавая работающий код.

Даже после того, как у вас соберется небольшая коллекция XML-файлов, может случиться так, что вы по-прежнему будете испытывать желание внести изменения в ваше определение DTD. Проблема заключается в том, что при этом вам неизбежно придется изменить и программу, и XML-файлы, чтобы обеспечить соответствие с определением DTD. Вносить изменения в программу нетрудно, если заранее решить, что все зависящее от XML будет помещаться в один Java-класс. Имейте в виду: при внесении изменений в определение DTD очень удобно использовать инструмент глобального поиска и замены. Например, в первой версии своего определения DTD мною для разметки языка был использован атрибут `"LANG"` вместо стандартного атрибута `"xml:lang"`, и я уже написал тридцать два XML-файла, используя старое определение. При помощи комбинации скриптов оболочки UNIX и команды `sed` мне удалось за минуту осуществить замену `"LANG"` на `"xml:lang"`. Внести изменения в определение DTD нетрудно, пока проект ограничен вашим компьютером, но когда в него вовлечено много людей, файлов и приложений, задача становится практически невыполнимой; так что тщательно шлифуйте свое определение DTD на ранних стадиях развития проекта.

Короче говоря, если вы определяете свой собственный формат файла, использующего XML, вам придется писать свое собственное определение DTD. Приступая к этой работе, я прежде всего представил, как должен выглядеть мой XML-файл. Конечно, формат для названия семейства, латинского и народного названия растений должен быть жестким, как в базах данных, а вот формат для текстового описания растений может быть свободным, подобным HTML. Я писал свое определение DTD постепенно, добавляя элементы, атрибуты, и компоненты по одному за раз и одновременно менял интерпретирующую программу, поскольку такой подход позволяет легко осваивать XML.

Обработка XML-документа

Трехуровневая архитектура

Чем особенно нравится мне процесс построения подобного приложения, так это широкими возможностями опробовать при его создании новые идеи. В конце концов, если все время делать одно и то же, вряд ли можно надеяться стать квалифицированным программистом. Приступая к работе, я первым делом обдумал общий план проекта и порядок, которого, по моему твердому убеждению, необходимо придерживаться до мелочей.

Основное архитектурное решение заключалось в том, что программа должна быть организована на трех уровнях: ввода, данных, вывода.

Уровень данных – это набор Java-классов и объектов программы, в которых будут храниться описания растений; содержаться они должны между уровнями ввода и вывода. Работа уровня ввода заключается в преобразовании XML-элементов, выдаваемых анализатором MSXML, в формат уровня данных. На уровне вывода идет преобразование информации, находящейся в уровне данных, в HTML-документы.

Основное соображение, благодаря которому я решил воспользоваться принципом трех уровней, заключалось в том, что XML является быстро развивающейся технологией, в которой я еще не до конца разобрался. Если формировать программу вокруг анализатора MSXML, многие решения пришлось бы передоверить разработчику этого анализатора. Другими словами, не имея опыта работы с MSXML, я невольно дал бы подвести себя к целому ряду решений, которые так бы и остались для меня тайной за семью печатями. Поэтому я решил идти своим путем и сконструировать приложение таким образом, чтобы от анализатора MSXML зависел только один класс. В таком случае, если бы мне потребовалось использовать другой XML-анализатор, или я решил отказаться, например, от XML в пользу базы данных, эту операцию можно было бы легко осуществить. Написать уровень вывода значительно проще, потому что он распознает только специальные объекты из уровня данных, а ведь я

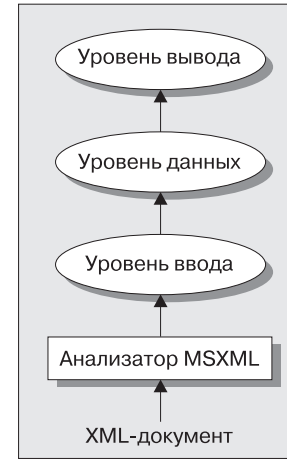


Рис. 12.1. Трехуровневая структура проекта

как раз и хотел овладеть *методами*, но не созданием коллекции элементов, атрибутов и компонентов, описывать которые в виду их уникальности всегда сложно. И, наконец, если бы мне пришло в голову не выводить HTML, а скажем, распечатать описание флоры деревни Эль Лимон при помощи редактора LaTeX, я мог бы не трогать уровни данных и ввода, а просто написать новый уровень вывода.

Другая причина, по которой работа над уровнем данных показалась мне полезной, – это строго типизированное написание в языке Java. Можно непосредственно превратить элементы, возвращаемые анализатором MSXML, в HTML. Например, я мог бы получить сведения о языке элемента `text`, выполняя

```
msxmlElement.getAttribute("xml:lang");
```

вместо

```
myTextObject.getLanguage().
```

Но предположим, что я ошибся в написании имени атрибута. Если бы я зафиксировал

```
msxmlElement. getAttribute("xml:fang")
```

программа все равно была бы скомпилирована, и анализатор MSXML просто вернул бы пустое сообщение в ответ на вызов этого метода. Если не вставить код, детектирующий подобное нулевое сообщение, неверный результат может проникнуть в программу и привести к непредсказуемым последствиям, включая такие, какие было бы невозможно предугадать до тех пор, пока конечный пользователь не начнет работать с программой. С другой стороны, если бы я ввел:

```
myTextObject.getFangluage()
```

Java-компилятор отреагировал бы мгновенно. Поместив весь код, касающийся XML, в одно место, я минимизировал число необходимых обращений к атрибутам по имени, а значит, и риск допустить ошибку такого рода. Самым важным при построении программы для меня являлась простота – это было принципиальное требование. Я хотел сделать свой Web-сайт максимально быстро, сохранив при этом возможность его расширения. Мне не хотелось тратить время и усилия на развитие возможностей, в которых не было нужды. И, наконец, чем проще программа, тем менее объемный код придется писать, тем быстрее будет закончена работа.

Я использовал механизмы компоновки и управления доступом языка Java, чтобы облегчить построение трехуровневой архитектуры. Хотя можно самому придумать конфигурацию и построить ее при помощи очень строгих правил, приятным, а главное, полезным, свойством строго записываемого языка является возможность заставить сам компилятор проводить в жизнь конструктивные решения.

Обратная сторона подобного решения заключается в том, что на войну с компилятором можно потратить много времени, однако, если работать, согласуясь с языком, а не бороться с ним, скоро обнаруживается, что строгое написание помогает разработчику самодисциплинироваться. Но какая именно дисциплина здесь нужна? В чем она должна была выразиться? Прежде всего, у меня было ясное представление, как соединить данные уровней данных и вывода, и я был готов

формализовать отношения между этими уровнями. С другой стороны, у меня не было уверенности в том, каким должно быть соотношение между уровнями данных и ввода; мне, например, хотелось бы иметь возможность легко его изменять.

В связи с этими соображениями было принято решение разбить программу на два разных Java-пакета: один, `honeylocust.limon`, содержащий уровень вывода, и другой, `honeylocust.limon.representation`, содержащий уровни ввода и данных. При таком подходе Java обеспечивает строгий контроль над интерфейсом уровня данные-вывод и более слабый контроль над интерфейсом уровня ввод-данные. Все эти манипуляции не означали, что меня не интересовала хорошая конструкция для интерфейса уровня ввод-данные. Я просто решил поэкспериментировать с разными идеями и открыть некоторые перспективы на будущее, прежде чем зафиксировать определенное решение.

Уровень данных

Объект Species.java

Объектом, используемым для представления растения, является `Species`. Поскольку моей целью была перспектива замены различных реализаций объекта `Species` (так чтобы можно было заново перенумеровывать растения), я решил написать для него и интерфейс, и реализацию. Интерфейс содержит все методы, которые хотелось бы сделать доступными уровню вывода. Из этого интерфейса следовало изъять методы, которые могут устанавливать или изменять состояние объекта `Species`. Уровень вывода не может менять объект `Species`, то есть объект `Species` является раз и навсегда установленным по отношению к уровню вывода.

Когда я только начинал строить объектно-ориентированные системы, у меня сложилось ощущение, что необходимо встраивать максимальный уровень функциональных возможностей в каждый разрабатываемый класс, так что я всегда вводил в свои объекты методы, которые позволяли бы другим объектам их изменять. Впоследствии я понял, что когда требуются только необходимые методы, можно писать менее громоздкие коды и при этом разработчик оказывается в некотором выигрыше. Например, объекты, существующие в многопоточном окружении, таком как GUI-системы (скажем, Java AWT и Microsoft Windows являются поточными системами) и серверные платформы, могут быть повреждены, когда два потока одновременно пытаются их изменить. Неизменный объект невосприимчив к подобным случаям, известным как «гонки», поскольку он вообще не может быть изменен. Если мы хотим модифицировать «неизменный» объект, мы просто создаем новый объект с теми исправлениями, которые хотели внести. Поскольку предыдущий объект сохраняется, все добавления легко отозвать, если мы хотим реализовать такие возможности, как многоуровневая отмена действий или транзакции.

Кроме того, чем меньше и проще интерфейс, тем менее громоздкий код приходится писать и обслуживать.

```
package honeylocust.limon.representation;

import java.io.File;

public interface Species {
```

```
    public String getId();
    public LanguageString getFamily();
    public LanguageString[] getLatin();
    public LanguageString[] getCommon();
    public Text[] getTexts();
    public boolean identified();
    public File getBigImage();
    public File getSmallImage();
};
```

Класс SpeciesImpl.java

Класс `SpeciesImpl` является реализацией объекта `Species`. Поскольку `SpeciesImpl` не является *общедоступным* (`public`) классом, невозможно создать экземпляр этого класса при помощи кода в пакете, отличном от `honeylocust.limon.representation`. А поскольку он реализует интерфейс `Species`, то доступ к методам, определенным в `Species`, может быть осуществлен только из уровня вывода, и больше ниоткуда. Обратите внимание, что при просмотре с уровня ввода этот объект не является неизменным. Если бы я хотел, чтобы он остался неизменным, я мог бы создать конструктор, содержащий все данные в этом объекте, и сделать все поля *закрытыми* (private). Однако такой конструктор является слишком сложным механизмом и на ранней стадии разработки кода потребовал бы частого внесения изменений, так что я решил позволить только классу уровня ввода, а именно классу `MSXMLSpeciesFactory`, создавать объект `Species` и инициализировать его, внося записи непосредственно в поля объекта `SpeciesSpecies`.

В этом классе проявилось одно из общих правил, которых я придерживался в свой работе; оно касается дисциплины при написании кода. При программировании на Java очень легко запутаться, какие переменные являются *локальными* (local), а какие полями класса. Чтобы покончить с этой путаницей, я взял привычку начинать все имена переменных полей данных с `"d_"`. Все мои статические переменные (которые совместно используются всеми экземплярами класса) начинаются с `"s_"`, а реконфигурируемые параметры с `"p_"`. Это соглашение не обязательно, но с тех пор, как я стал придерживаться такого правила, мне удалось избавиться от множества проблем.

```
package honeylocust.limon.representation;

import java.util.Vector;
import java.io.File;

class SpeciesImpl implements Species {

    String d_id;
    LanguageString d_family;
    LanguageString d_latin[];
    LanguageString d_common[];
    Text d_texts[];
    File d_bigImage;
    File d_smallImage;
```

```
    public String getId() {
    return d_id;
    };

    public LanguageString getFamily() {
    return d_family;
    };

    public LanguageString[] getLatin() {
    return (LanguageString[]) d_latin.clone();
    };

    public LanguageString[] getCommon() {
    return (LanguageString[]) d_common.clone();
    };

    public Text[] getTexts() {
    return (Text[]) d_texts.clone();
    };

    public boolean identified() {
    if (d_latin==null)
        return false;

    if (d_latin.length==0)
        return false;

    return true;
    };

    public File getBigImage() {
    return d_bigImage;
    };

    public File getSmallImage() {
    return d_smallImage;
    };

};
```

Класс LanguageString.Java

`LanguageString` – это класс, представляющий строку и язык, на котором написана эта строка. `LanguageString` является примером полностью неизменяемого класса. Интернирование строки – это скрытая возможность языка Java. Оно будет использоваться в классе `LanguageString`, как, впрочем, и в других случаях. Я познакомился с интернированием на http://developer.netscape.com/docs/technote/simple/Internment.html.

В классе `String` языка Java есть метод `String.intern()`, возвращающий ссылку на строку, которая вызвана с его помощью. Каждая ссылка уникальна, но ссылается на одну и ту же строку. То есть каждая из интернированных строк с одинаковым значением является одной и той же строкой. Рассмотрим следующий участок кода:

```
String a="Hello";
String b="Hello";

String c=a.intern();
String d="Hello".intern();

if (a==b)
   Sysem.out.println("a==b");

if (c==d)
   System.out.println("c==d");
```

Вывод для показанного участка будет такой:

```
<output>
c==d
</output>
```

Такой способ работает потому, что объект `String` аналогичен любому другому объекту. На самом деле `"a"` и `"b"` – это ссылки на два различных, независимо созданных объекта `String`. Оператор равенства для объектов сравнивает ссылки на объекты: два объекта будут считаться равными, если являются в точности одинаковыми объектами, а не в случае равенства *значений* этих объектов. Поскольку `a` и `b` указывают на две разные строки, выражение `a==b` ложно.

Безусловно, всегда можно сравнивать значения, используя `String.compareTo()` или `String.eduals()`, но иногда удобнее сравнивать строки при помощи оператора `==`. Именно в этот момент на сцену выходит интернирование. Метод `String.intern()` использует хэш-таблицу для отслеживания каждой строки, которую он создает. Таким образом, он возвращает только одну строку с любым конкретным значением. Поскольку объекты `c` и `d` оба имеют значение `"Hello"` и оба были созданы при помощи интернирования, объекты `c` и `d` являются одинаковыми объектами, и `c==d` истинно.

При использовании интернирования мне пришлось пойти на компромисс. Интернирование строки требует некоторого времени, но впоследствии удается легче осуществлять сравнение строк и быстрее писать программу. Каждая отдельная строка, которую вы интернируете при помощи `intern()`, занимает память во внутренней хэш-таблице, которая никогда не будет освобождена программой «сборки мусора», так что потенциально это приводит к излишним затратам ресурсов. Хорошо проводить интернирование строк, являющихся символами, а также строк, часто имеющих одинаковые значения (такие как ключевые слова), и имен переменных на языке программирования. Например, стоит интернировать строки, которые представляют собой названия (человеческих) языков (чтобы быть точным, строки ISO 639), поскольку много разных строк `LanguageStrings` будут написаны на одних и тех же немногих языках. В этом случае моя программа не поглощает много памяти и времени, так что я использовал интернирование, чтобы легче записывать сравнение строк при разработке кода; ведь время, затраченное на разработку, часто обходится дороже, чем компьютерное время.

Интернирование – это не только то, что Java делает для строк (`Strings`), но и то, что вы можете сделать для своих собственных классов. Класс `LanguageString` сам по себе предоставляет возможность интернирования. Я встроил его, поскольку многие растения относятся к одинаковым семействам (то есть имеют одинаковые

названия семейств), и с помощью этого класса удобнее проверять, относятся ли растения к одному семейству. Автор анализатора MSXML также решил, что интернирование хорошо подходит для объектов `Name`. Поскольку не существует *общедоступного* (public) конструктора для объектов `Name`, каждый экземпляр создается через функцию `Name.create()`, так что все объекты `Name` интернированы.

Другой особенностью JDK 1.1, использованной в `LanguageString`, является `Blank Finals`. Когда вы записываете что-то вроде:

```
private final String d_string;
```

вы описываете переменную `final` (и значит, неизменяемую). В JDK 1.0.2 переменная `final` работала так же, как `const` в языке C, и значение переменной должно было быть указано в момент ее описания, наподобие

```
private final String d_string="foo";
```

В JDK 1.1 вы не обязаны помещать значение `final` переменной в место ее описания; предполагается, что вы укажете ее значение в статическом инициализаторе и во всех конструкторах и после этого никогда не будете его менять.

```
package honeylocust.limon.representation;

import com.sun.java.util.collections.Comparable;
import java.util.Hashtable;

public class LanguageString implements Comparable {

    private final String d_string;
    private final String d_language;

    private static String s_defaultLanguage="en".intern();
    private static final Hashtable s_symbols=new Hashtable();
            // Используется для свойства intern().

    public LanguageString(String s) {
        d_string =s;
        d_language=s_defaultLanguage;
    };

    public LanguageString(String s,String lag) {
        d_string=s;
        if (lang==null)
        lang=s_defaultLanguage;

        d_language=lang.intern();
    };

    public String toString() {
        return d_string;
    };

    public String getLanguage() {
        return d_language;
    };

    public int compareTo(Object o) throws ClassCastException {
```

```
    if (!(o instanceof LanguageString))
        throw new ClassCastException();

    LanguageString other=(LanguageString) o;

    // Следующий фрагмент - потенциальное узкое место, но пока он работает.

    return (d_string.toUpperCase()).compareTo(other.d_string.toUpperCase());
    };

// Для LanguageString эта часть кода делает то же самое, что String.intern() ..
// ..делает для String.

    public LanguageString intern() [
    String uniqname=d_string+"&"+d_language;
    synchronized(s_symbols) {
        LanguageString interned=(LanguageString) s_symbols.get(uniqname);
        if (interned==null) {
        interned=this;
            s_symbols.put(uniqname,this);
        }
        return interned;
    }
    }
};
```

Интерфейс Text.java и класс TextImpl.java

Интерфейс `Text` – это отрывок текста, отмеченный так за свое происхождение и язык. Он соответствует элементу `<text>` в `limon.dtd`. `Text` – это совокупность текстовых участков (`TextChunks`). Поскольку я непосредственно заинтересован в обеспечении альтернативной реализации `Text`, я создал интерфейс и обеспечил `TextImpl.java`, который является его реализацией.

```
package honeylocust.limon.representation;

public interface Text {

    public String getType();
    public TextChunk[] getChunks();
    public String getSource();

    public String getLanguage();
    public String getText();
};

package honeylocust.limon.representation;

class TextImpl implements Text {

    TextChunk[] d_chunks;

    String d_type;
    String d_source;
    String d_language;

    public TextImpl() {
    };
```

```
    public TextImpl(Text t) {
    d_type=t.getText();
    d_source=t.getSource();
    d_language=t.getLanguage();
    d_chunks=t.getChunks();
    };

    public String getType() {
    return d_type;
    }

    public TextChunk[] getchunks() {
    return ((TextChunk[]) d_chunks.clone());
    }

    public String getSource() {
    return d_source;
    }

    public String getLanguage() {
    if (d_language==null)
    return "en".intern();

    return d_language;
    }

    public String getText() {
    StringBuffer sd=new StringBuffer();

    for(int i=0; i<d_chunks.length; i++)
        sb.append(d_chunks[i].getText());
    return sb.toString();
    };
e
};
```

Интерфейс TextChunk.java

`TextChunks` – это интерфейс сам по себе. Единственная его задача – вернуть представление по умолчанию самого себя в виде строки. Подобная операция дает нам возможность распечатывать `Text`, просто сцепляя все строки вместе. Более отлаженная программа для вывода текста могла бы предоставить свой код для осуществления каких-то специальных операций с `RefChunks`, `AnchorChunks` и другими специальными типами *участков* (chunk). Поскольку я не хотел встраивать в этот код что-либо относящееся к HTML, разнообразные типы участков просто возвращают информацию о себе, а не HTML-код для их представления.

```
package honeylocust.limon.representation;

import java.util.Vector;

public interface TextChunk {
    abstract public String getText();
};
```

Класс PlainChunk.java

```
package honeylocust.limon.representation;

public class PlainChunk implements TextChunk {

    private final String d_text;

    public PlainChunk(String text) {
    d_text=text;
    };

    public String getText() {
    return d_text;
    };
};
```

Класс AnchorChunk.java

```
package honeylocust.limon.representation;

public class AnchorChunk implements TextChunk {

    String d_text;
    String d_href;

    public AnchorChunk(String text,String href) {
    d_text=text;
    d_href=href;
    };

    public String getText() {
    return d_text;
    }

    public String getHref() {
    return d_href;
    };

};
```

Класс CMChunk.java

```
package honeylocust.limon.representation;

public class CMChunk implements TextChunk {

    String d_text;

    public CMChunk(String text) {
    d_text=text;
    };

    public String getText() {
    return d_text+"cm";
    };

};
```

Интерфейс RefChunk.java

```
package honeylocust.limon.representation;

public interface RefChunk extends TextChunk {
```

```
    public String getId();
};
```

Класс RefChunkImpl.java

```
package honeylocust.limon.representation;

public class RefChunkImpl implementRefChunk {

    private final String d_id;

    public RefChunkImpl(String id) {
    d_id=id;
    };

    public String getText() {
    return "Plant "+d__id;
    };

    public String getId() {
    return d_id;
    };

};
```

Заключение

Поскольку оба языка, XML и HTML, развиваются очень быстро, я не хотел, чтобы конструкции обеих моих программ – читающей XML и создающей HTML – были связаны воедино. Поэтому было решено создать уровень данных, обеспечивающий представление описаний растений вне зависимости от деталей программы, читающей XML, и от деталей программы, создающей HTML. Структура уровня данных параллельна структуре моих XML-файлов. Например, объект `Species` соответствует тэгу `<species>`, объект `Text` соответствует тэгу `<text>`. Доступ к атрибутам и более простым дочерним элементам теперь осуществляется посредством методов. Например, доступ к народным названиям теперь можно получить, применив метод `getCommon()` объекта `Species`. Имена представляются объектами `LanguageString`, которые содержат и строку, и буквенный код, характеризующий язык, на котором написана эта строка (например, `"en"` для английского).

Уровень ввода

Доработка исходного кода MSXML

В соответствии со спецификацией XML определение типа документа (DTD) может включать в себя текст из другого файла через внешний компонент, почти так же как С-программа включает в себя текст посредством директивы `#include`. Удивительно, но анализатор MSXML версии 1.8 на языке Java не предусматривает подобный вариант, поэтому мне пришлось внести небольшие изменения в исходный код. Перед тем, как добавить эту характеристику, я решил просмотреть лицензионные ограничения на производимые работы, но, к счастью, оказалось, что Microsoft беспокоилась главным образом о том, чтобы пользователи не пытались представить свою модифицированную версию как исходный продукт Microsoft.

Чтобы избежать путаницы, которая может возникнуть в том случае, когда люди и программы захотят использовать первоначальную версию анализатора MSXML, я переименовал пакеты MSXML Java. Это было проделано при помощи автоматического поиска и замены строки `"com.ms.xml"` на строку `"honeylocust.msxml"` во всех файлах анализатора MSXML.

Часто дефекты и неполадки в программе могут быть исправлены в результате изменения одной строки. Задача состоит в том, чтобы найти, в какую из 11080 строк, созданных кем-то другим, следует внести изменения. Изучение чужого кода сравнимо с разглядыванием через замочную скважину – наблюдая в определенный момент только маленькую часть программы, необходимо суметь разобраться во всем ее массиве, чтобы найти нужный фрагмент, в который необходимо внести изменения.

Первым признаком, вызвавшим беспокойство, было сообщение:

```
<output>
Missing entity 'aacute'
Location: file:/afs/msc.cornell.edu/home/henley/hoile/media/limon/plantxml/10.
xml(13,38)
Context: <PLANTDATA><SPECIES><COMMON>
</output>
```

Когда я скопировал описание внутреннего компонента непосредственно в свое определение DTD, все сработало правильно, поэтому можно было сделать вывод, что в мое определение DTD не были загружены внешние компоненты. Просмотрев документацию, я заметил, что объект `Document` имеет метод `"setLoadExternal()"`, который управляет загрузкой внешних компонентов. Я нашел в коде этот метод:

```
public void setLoadExternal(boolean yes) {
   loadExternal = yes;
```

и сообразил, что объект `Document` имеет поле, называемое `loadExternal`. Я использовал команду `grep` в UNIX, чтобы найти каждое появление слова `"loadExternal"` во всем источнике и получил:

```
<output>
om/Document.java:   loadExternal = true;
om/Document.java:   loadExternal = true;
om/Document.java:   loadExternal = yes;
om/Document.java: public boolean loadExternal()
om/Document.java:   return loadExternal;
om/Document.java:   parser.parse(url,this,dtd,this,caseInsensitive,loadExternal);
om/Document.java:   parser.parse(in,this,dtd,this,caseInsensitive,loadExternal);
om/Document.java: boolean loadExternal;
</output>
```

Ссылки на `loadExternal` были только в методе `Document`, но значение `loadExternal` передавалось в анализирующий метод анализатора.

```
public void load(URL url) throws ParseException {
   clear();
   URLdoc = url;
```

```
        parser = new Parser();
        parser.parse(url,this,dtd,this,caseInsensitive,loadExternal);
    }
```

Это заставило меня сконцентрировать внимание на объекте `Parser` в пакете `com.ms.xml`. Я просмотрел метод `parse`:

```
    public final void parse(URL url, ElementFactory factory, DTD dtd, Element
            root, boolean caseInsensitive, boolean loadExt) throws
            ParseException
    {
        this.dtd = dtd;
        this.root = root;
        this.loadexternal = loadExt;
        setURL(url);
        setFactory(factory);
        this.caseInsensitive = caseInsensitive;
        safeParse();
    }
```

и обнаружил, что данный анализатор имеет свое собственное поле, называемое `"loadexternal"`, которое контролирует, были ли загружены внешние компоненты. Далее просмотрел, как используется эта переменная, и нашел:

```
    if (loadexternal)
        loadDTD(en.getURL(), current.defaultNameSpace);
```

Эта запись и привела меня к решению проблемы. Выяснилось, что в методе `loadDTD` MSXML создает новый анализатор, чтобы проанализировать определение DTD. Единственная проблема заключалась в следующем: новому анализатору не сообщается, что он тоже должен загрузить внешние компоненты. Подобное расширение заняло всего одну строку:

```
    public final void loadDTD(String urlStr, Atom nameSpace) throws ParseException {
        try {
            URL u = new URL(url, urlStr);
            Parser parser = new Parser();
            parser.dtd = this.dtd;
            parser.setURL(u);
            parser.setFactory(factory);
            parser.caseInsensitive = this.caseInsensitive;
            parser.loadexternal=this.loadexternal; // Добавил Paul Houle 1998.
            Element root = factory.createElement(null,Element.ELEMENT,
    nameDOCTYPE,null);
            parser.newContext(root, null, Element.ELEMENT, false, nameSpace,
            current.spaceTable);
            parser.parseInternalSubset();
        }   catch (IOException e) {
            error("Couldn't find external DTD '" + urlStr + "'");
        }
    }
```

Эта строка заставляет новый анализатор наследовать то же значение `loadExternal`, которое имеет текущий анализатор. Теперь внешние компоненты могут использоваться внутри определения DTD.

Уровень ввода: превращение XML в Species

Для того, чтобы иметь возможность сразу приступить к работе, я решил поместить все имеющее отношение к MSXML в один класс. Это значило, что при желании или необходимости переключиться на другой анализатор или формат данных, класс просто следует поменять. К тому же, когда разработчик вносит исправления в свое определение DTD, или в случае, если вводят изменения в код, который имеет отношение к XML, – существует только один фрагмент, который следует доработать.

Класс MSXMLSpeciesFactory

`MSXMLSpeciesFactory` – это производящий класс, и все его методы статические. На него, как объявлено в определении DTD, возложена обязанность создания и полной инициализации новых экземпляров `Species`. Сначала мы импортируем все необходимые классы из MSXML и из стандартных библиотек Java. Хотя можно импортировать пакеты целиком за один раз. Это выполняется следующим образом:

```
import honeylocust.limon.msxml.om.*;
```

Однако импортируя классы порознь, я хорошо вижу, от чего зависит мой код. Это также помогает читающим код понять, что с чем связано.

```
package honeylocust.limon.representation;

import honeylocust.msxml.om.Document;
import honeylocust.msxml.om.Element;
import honeylocust.msxml.om.ElementCollection;
import honeylocust.msxml.parser.ParseException;
import honeylocust.msxml.util.Name;

import java.io.File;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Vector;
```

Затем создаются несколько статических полей, которые позволяют компактным образом ссылаться на многократно встречающиеся имена. Проще писать `n_Species`, чем `Name.create("species")`. Подобный стиль привлекает компилятор к проверке орфографии имен (Names). Как и в случае использования приставки `"d_"` для обозначения полей данных, я начинаю имена с `"n_"`, чтобы в дальнейшем их было легче отслеживать.

```
public class MSXMLSpeciesFactory {

    static final Name n_SPECIES=Name.create("species");
    static final Name n_FAMILY=Name.create("family");
    static final Name n_LATIN=Name.create("latin");
    static final Name n_COMMON=Name.create("common");
    static final Name n_TEXT=Name.create("text");
```

```
    static final Name n_A=Name.create("a");
    static final Name n_CM=Name.create("cm");
    static final Name n_REF=Name.create("ref");
...
```

MSXMLSpeciesFactory имеет два общедоступных (public) метода. Один принимает файл и аргумент, другой – URL. Оба считывают тэги <species> и их дочерние тэги из XML и добавляют их к вектору. MSXML может загружать XML из одного или из двух источников, а именно, из потока ввода или из URL. Хотя возможность передать FileInputStream в MSXML существует, но в этом случае MSXML не будет знать, в какой директории находится данный файл. Это важное обстоятельство, потому что мне хотелось бы, чтобы MSXML находил определение DTD, помещенное в той же директории, что и XML-файл. Ссылка на определение DTD при помощи относительного URL решает проблему. Версия parseLimon, которая принимает файл, всего лишь конвертирует объект File (на самом деле он представляет собой имя файла в кросс-платформном написании) в URL файла, а затем вызывает ту версию parseLimon, которая принимает URL.

Эта версия создает новый объект XML-документа, конфигурирует анализатор так, чтобы позволить произвести загрузку внешних объектов, таких как определение DTD, и использует метод Document.load(), чтобы загрузить документ.

Документ, согласующийся с limon.dtd, содержит единственный элемент <plantdata>, который включает несколько элементов <species>. Мы получаем корневой элемент, и затем используем root.getChildren(), чтобы получить коллекцию ElementCollection элементов SpeciesElements, которую мы передадим нашему собственному parseSpecies для обработки.

```
public static void parseLimon(Vector v,File f) throws Exception {
parseLimon(v,new URL("file:"+f.getAbsolutePath()));
};

public static void parseLimon(Vector v,URL u) throws Exception {
Document d=new Document();
d.setLoadExternal(true);
try {
   d.load(u);
}      catch(ParseException e) {
   d.reportError(e,System.out);
   System.exit(-1);
};

Element root=d.getRoot();
ElementCollection plants=root.getChildren();

for(int i=0;i<plants.getLength();i++) {
   Element plantElement=plants.getChild(i);
   if (plantElement.getTagName()==n_SPECIES) {
           Species species=MSXMLSpeciesFactory.parseSpecies(plantElement);
           v.addElement(species);
   };
};
};
```

Один из самых простых способов организации класса заключается в описании метода для анализа каждого типа элемента, `parseSpecies`, который принимал бы объект `Element` как параметр и возвращал объект `Species`. При этом необходимо проводить проверку, действительно ли переданный объект `Element` является элементом `Species`. Поскольку MSXML возвращает `NULL` при попытке обратиться к несуществующему элементу или атрибуту, использование «оборонительного» стиля программирования представляется хорошей идеей.

Итак, мы создаем `SpeciesImpl`, который располагается по дереву ниже `Species` после выхода из данного метода. Поскольку метод, которому известно только, что объект принадлежит типу `Species`, не способен неверно обращаться с его полями по умолчанию, это укрепляет контроль доступа. Конечно, такая мера не может служить несокрушимой защитой от программиста, намеренного нарушить данное условие, но с помощью этого метода все-таки можно четче представить правила использования, принятые внутри пакета.

Затем проходим по атрибутам и подэлементам элемента `<species>`, чтобы проанализировать его. Сначала, используя `Element.getAttribute()`, захватываем атрибут `ID`. Поскольку элемент семейства является простым, включаем код для его анализа внутрь метода. Следует учесть, что мы интернируем `LanguageString` при помощи `intern()`, потому что многие растения принадлежат одному семейству и куда проще проводить сравнение непосредственно самих объектов.

Полезно также создавать удобные методы для функций, которые используются по несколько раз. Например, метод `getSingleChild()` позволяет легко справляться с ситуациями, когда, как в случае с `<family>`, находящимся внутри `<species>`, в элементе имеется не более одного подэлемента, и вы не хотите путаться с содержанием `ElementCollection`.

```
    static Species parseSpecies(Element e) throws Exception {
    if (e.getTagName()!=n_SPECIES)
        throw new Exception("Element must be a SPECIES Element for second-stage
parse!");

    SpeciesImpl species=new SpeciesImpl();

    species.d_id=(String) e.getAttribute("id");
    Element familyElement=getSingleChild(e,n_FAMILY);
    if (familyElement!=null)
        species.d_family=new LanguageString(familyElement.getText(),"la").intern();

    species.d_latin=convertToLanguageStringArray(e,n_LATIN,"la");
    species.d_common=convertToLanguageStringArray(e,n_COMMON);

    ElementCollection textElements=new ElementCollection(e,n_TEXT,-1);
    Text[] texts=new Text[textElements.getLength()];

    for(int i=0;i<textElements.getLength();i++) {
        texts[i]=parseText(textElements.getChild(i));
    };

    species.d_texts=texts;

    return species;
    };
```

```
    // Метод для удобства:
    // Если имеется единственный потомок данного типа, этот метод извлекает его..
    // ..не заботясь о Collection.

    static Element getSingleChild(Element e,Name kind) throws Exception {
    ElementCollection ec=new ElementCollection(e,kind,-1);
    if (ec.getLength()>1)
        throw new Exception("I'm only supposed to get a single child!");
    if (ec.getLength()==0)
        return null;
    else
        return ec.getChild(0);
    };

    /* В этом случае в XML нет языковых данных, и все.. */
    /* строки будут на одном языке. */

    static LanguageString[] convertToLanguageStringArray(Element e,Name kind,String
lang) throws Exception {
    ElementCollection ec=new ElementCollection(e,kind,-1);

    LanguageString[] sarray=new LanguageString[ec.getLength()];

    for(int i=0;i<ec.getLength();i++) {
        sarray[i]=new LanguageString(ec.getChild(i).getText().trim(),lang);
    };

    return sarray;
    };
```

Кроме того, `convertToLanguageStringArray` обходит дефект в MSXML 1.8. По-видимому, MSXML не отслеживает автоматически значения атрибутов по умолчанию, так что приходится заполнять их самостоятельно.

```
    static LanguageString[] convertToLanguageStringArray(Element e,Name kind)
throws Exception {
    ElementCollection ec=new ElementCollection(e,kind,-1);
    LanguageString[] sarray=new LanguageString[ec.getLength()];

    for(int i=0;i<ec.getLength();i++) {
        Element child=ec.getChild(i);
        String lang=(String) child.getAttribute("xml:lang");
        if (lang==null)
            lang="en";

        sarray[i]=new LanguageString(ec.getChild(i).getText().trim(),lang);
    };

    return sarray;
    };
```

Аналогично тому, как `parseSpecies` анализирует `<species>`, метод `parseText()` работает с тэгом `<text>`.

```
    static Text parseText(Element e) {
    TextImpl text=new TextImpl();

    text.d_type=(String) e.getAttribute("type");
```

```
    text.d_source=(String) e.getAttribute("source");
    text.d_language=(String) e.getAttribute("language");

    ElementCollection chunkElements=new ElementCoolection(e,null,-1);
    text.d_chunks=new TextChunk[chunkElements.getLength()];

    for(int i=0;i<chunkElements.getLength();i++) {
        Element f=chunkElements.getChild(i);

        if (f.getTagName()==n_A) {
            text.d_chunks[i]=parseAnchorChunk(f);
        } else if (f.getTagName()==n_CM) {
            text.d_chunks[i]=parseCMChunk(f);
        } else if (f.getTagName()==n_REF) {
            text.d_chunks[i]=parseRefChunk(f);
        } else
            text.d_chunks[i]=parsePlainChunk(f);

    };
    return text;
    };
```

И наконец, каждый тип текстового участка (text chunk) анализируется своим собственным методом.

```
    static PlainChunk parsePlainChunk(Element e) {
    return new PlainChunk(e.getText());
    };

    static CMChunk parseCMChunk(Element e) {
    return new CMChunk(e.getText());
    };

    static RefChunk parseRefChunk(Element e) {
    return new RefChunkImpl((String) e.getAttribute("ID"));
    };

    static AnchorChunk parseAnchorChunk(Element e) {
    return new AnchorChunk(e.getText(),(String) e.getAttribute("href"));
    };
};
```

Присоединение изображений

Имена файлов крупного и маленького изображений, прилагаемые к каждому описанию растения – это важные атрибуты элемента `Species`, которые не выставлены классом `MSXMLSpeciesFactory`. Поскольку эта информация в настоящее время не хранится в XML-файлах и не имеет никакого отношения к XML-анализатору, я решил организовать для данной функции собственный класс, `ImageAttach`. Экземпляру `ImageAttach` передается объект `File`, содержащий директорию, в которой будут размещены изображения. `ImageAttach` подразумевает, что в наличии будут иметься два файла, `bigImage` (большоеИзображение) с именем `[species id].gif` и `smallImage` (маленькоеИзображение) с именем `[species id]half.gif`. Этот класс

имеет важное значение, с его помощью можно удостовериться, что все изображения находятся на месте. Если файл изображения не найден, метод `attachImages()` выдает `FileNotFoundException`, вызывая преждевременное прекращение выполнения ввода.

Это, конечно, не самая красивая часть программы – здесь приходится пользоваться двумя методами, чтобы инициализировать `Species`. Сначала этот элемент создается классом `MSXMLSpeciesFactory`, затем к нему при помощи класса `ImageAttach` необходимо подключить изображение. Есть риск, что программист сделает первое, но забудет о втором. В больших проектах такая ошибка была бы опасной, но в нашем случае это всего лишь мелкий изъян.

```
package honeylocust.limon.representation;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Enumeration;
import java.util.Vector;
public class ImageAttach {

    final File d_path;
    d_path=path;
    };

    public void attachImages(Vector v) throws FileNotFoundException {
    for(Enumeration e=v.elements();e.hasMoreElements();) {
        SpeciesImpl s=(SpeciesImpl) e.nextElement();
        attachImages(s);
    };

    };

    void attachImages(SpeciesImpl s) throws FileNotFoundException {
    s.d_bigImage=safeFile(s.getId()+".gif");
    s.d_smallImage=safeFile(s.getId()+"half.gif");
    };

    File safeFile(String s) throws FileNotFoundException {
    File f=new File(d_path,s);
    if (!f.exists())
        throw new FileNotFoundException(s);

    return f;
    };
};
```

Заключение

Уровень ввода состоит всего из двух классов. Один из них – это `MSXMLSpeciesFactory`, который отвечает за использование MSXML для анализа XML-файлов и преобразования информации в объекты `Species`. Другой класс – `ImageAttach`, который находит файлы, содержащие изображения, убеждается, что таковые существуют, и завершает создание уровня данных, присоединяя изображения к объектам `Species`.

Уровень вывода

Конструирование HTML

Целью уровня вывода является преобразование объектов `Species` в Web-страницы, форматированные с применением стиля. Воспроизведение страниц в характерном и согласованном виде представлялось мне очень важной задачей, однако стоило мне всерьез задуматься над этой проблемой, как тут же возникли сомнения: что если жители Эль Лимона и другие пользователи пожелают просматривать мои страницы, используя программу автоматического перевода. Подобный вариант вносил существенные коррективы в мой проект. По адресу

http://babelfish.altavista.digital.com/

находится впечатляющая демонстрационная версия, которая автоматически переводит Web-страницы с английского на несколько европейских языков. Хотя файлы псевдошрифтового текста `.gif` способны существенно улучшить вид Web-страниц, я решил идти другим путем, поскольку псевдошрифтовый текст невозможно автоматически перевести.

Архитектура информации

Этот этап я начал с конструирования структуры сайта в целом, окончательный вид которого по существу отражает структуру печатных каталогов растений. Я решил, что у меня будет по одной Web-странице на каждое растение и страницы-указатели с индексами для латинских, народных названий, а также для наименований семейств. Безусловно необходима первая страница-«обложка», вводящая новых пользователей «Сорняков...» в приложение, и несколько дополнений, содержащих эссе о проекте и информацию об авторах Web-системы. Хотя большая часть структуры узла видна пользователю, мне также надо было решить, куда поместить все скрытые файлы, от которых зависит представление HTML, а именно, файлы с изображениями и таблицы стилей. На начальной стадии проекта структура директории выглядела, как показано на рис. 12.2.

Поскольку мне удалось построить издательскую систему, которая способна автоматически создавать сайт, заменить его структуру на более гибкую сложности не представляло.

Дело в том, что надо мной довлели три обстоятельства, заставляющие изменить организацию сайта. Первое – это ясное понимание, что нумерация растений не сообщает никакой информации, кроме обычного порядкового номера, присвоенного растению. Можно, конечно, рассчитывать, что мы оказались первооткрывателями самых распространенных в Центральной Америке видов флоры, но все это выглядело как-то несерьезно. Можно сказать, ненаучно... Во всех изданиях, описывающих растительный мир, его представители расположены по семействам, родам и разновидностям. При таком подходе то или иное растение отыскивается простым перелистыванием страниц, и навигационная система становится куда более приемлемой. Выходит, нам следовало изменить нумерацию в соответствии с общепринятой? Но если перенумеровывать каждый объект заново, то надо было бы также изменить очередность расположения иллюстраций, иначе HTML-файлы будут ссылаться на растения с другими номерами. К тому времени я только что

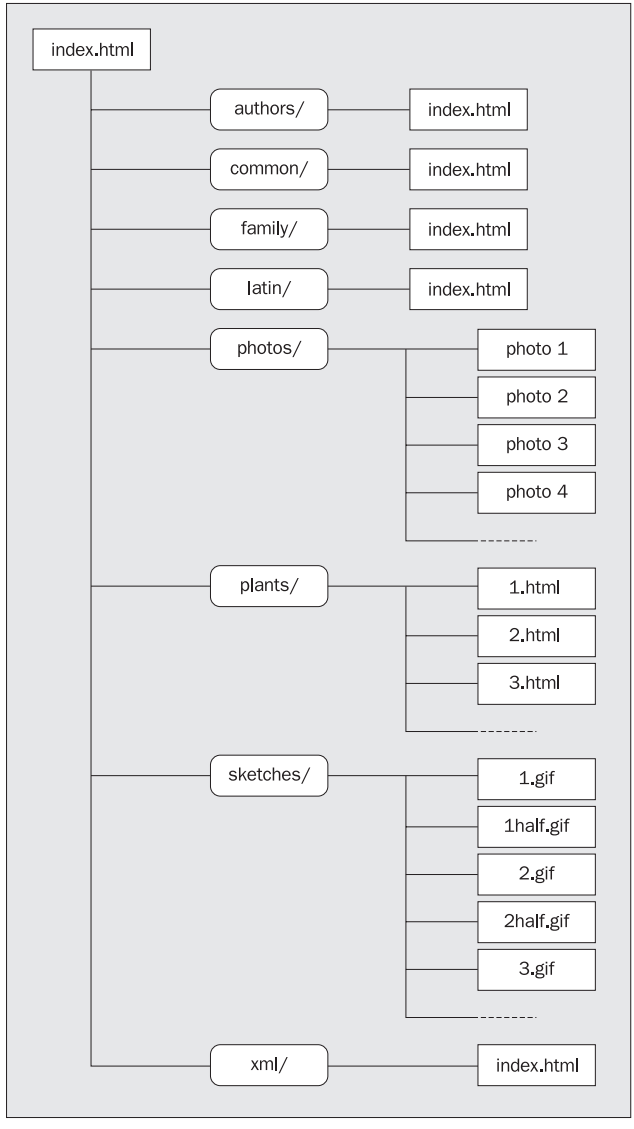


Рис. 12.2. Структура сайта на начальной стадии проекта

вручную скопировал каждую травинку в директорию «эскизы» (sketches), и XML-процессор их не касался; он просто подразумевал, что изображения в программе есть и на них можно устанавливать гиперссылки. Любое поспешное решение проблемы было чревато ошибками – если номера Web-страниц с описаниями растений отличались бы от номеров иллюстраций, страницы могли бы привести в замешательство того, кто осуществлял бы поддержку Web-сайта. Если уж придется заняться перенумерацией, этот процесс следовало бы автоматизировать, и

порядок растений (и их изображений) должен был бы изменяться, как только мы идентифицируем большее количество видов.

Вторым обстоятельством было то, что мы с Оливией прочитали пресловутую книгу Killer Web Sites («Убийца Web-сайтов»), написанную Дейвом Зейгалом (Dave Seigal) – с ней можно ознакомиться по адресу http://www.killersites.com/ – и это подвигло нас к мысли уделить больше внимания сжатию файлов с изображениями, что помогло бы добиться значительного ускорения загрузки узла без существенного ухудшения качества. Мы смогли почти вдвое уменьшить объем иллюстраций при незначительном ухудшении качества, уменьшив число цветов до 32 с помощью программы ImageMagick, которая находится по адресу:

http://www.wizards.dupont.com/cristy/ImageMagick.html

Продвигаясь в этом направлении, нам удалось переоформить наше издание таким образом, что оно, пусть даже и не выглядело теперь так хорошо, как первоначальный вариант, – уместилось на один гибкий диск, а это большой плюс для разработки, которая будет распространяться в технически слабо развитых районах. В итоге мы пришли к тому, что в нашем случае лучше всего предоставить выбор из различных наборов изображений. При этом появлялась возможность одновременно создать как стандартное издание с безупречными изображениями, так и компромиссный вариант, помещающийся на одном гибком диске.

Однако решающим оказалось последнее обстоятельство – именно оно стало последней каплей, которая заставила меня заняться модификацией приложения. На системах Unix работу по развитию и усовершенствованию сайта принято проводить на работающем Web-сервере, поскольку сервер Unix просто делает все файлы, расположенные ниже определенной директории, доступными всему миру. Чаще всего встречается вариант с двумя Web-серверами, один из них используется для тестирования, а второй для работы. Если вы не хотите показывать публике какую-то информацию, можно использовать парольную защиту. Редактирование узла на рабочем Web-сервере позволяет избежать проблем, возникающих из-за различий между конфигурациями сервера и локальной машины. В этом случае не существует риска, что файлы могут быть утеряны, записаны поверх нужных или искажены в момент их перемещения. Кроме того, все CGI скрипты и другие возможности сервера всегда находятся под рукой.

Читатели могут оценить степень разочарования, которое я испытал, когда пришло время выпускать издание «Сорняков...» на ZIP-диске для отправки в Эль Лимон. Оказалось, что в системе локальных файлов «Сорняки...» работают некорректно. Проблема заключалась в том, что я положился на сервер, который, как выяснилось потом, предоставляет страницы `"index.html"`, расположенные в директории, когда браузер запрашивает `"/"`. Таким образом, мне было необходимо изменить все мои ссылки, оканчивающиеся на `"/"`, на ссылки, оканчивающиеся на `"/index.html"`. Хотя я считал, что Web-страница приятнее смотрится, когда вся эта чужеродная для просмотра механика удалена, в продукте, предназначенном для людей, имеющих компьютер, но не имеющих подсоединения к сети, как в Эль Лимоне, важнее дать пользователям возможность работать с продуктом в локальной

файловой системе, а не на Web-сервере. Можно, конечно, установить Web-сервер на компьютере пользователя, однако намного проще так изменить расположение файлов, чтобы страницы правильно работали в локальных файловых системах.

Поскольку третья проблема безусловно требовала решения, заставляя меня переделать части, которые устанавливают гиперссылки, я решил, что у меня есть хороший шанс изменить организацию файлов. Задача состояла в том, чтобы собрать всю информацию в одном месте и сделать структуру понятной человеку, который будет работать с ней на Web-сайте. Сейчас общая структура приложения выглядит так, как показано на рис. 12.3.

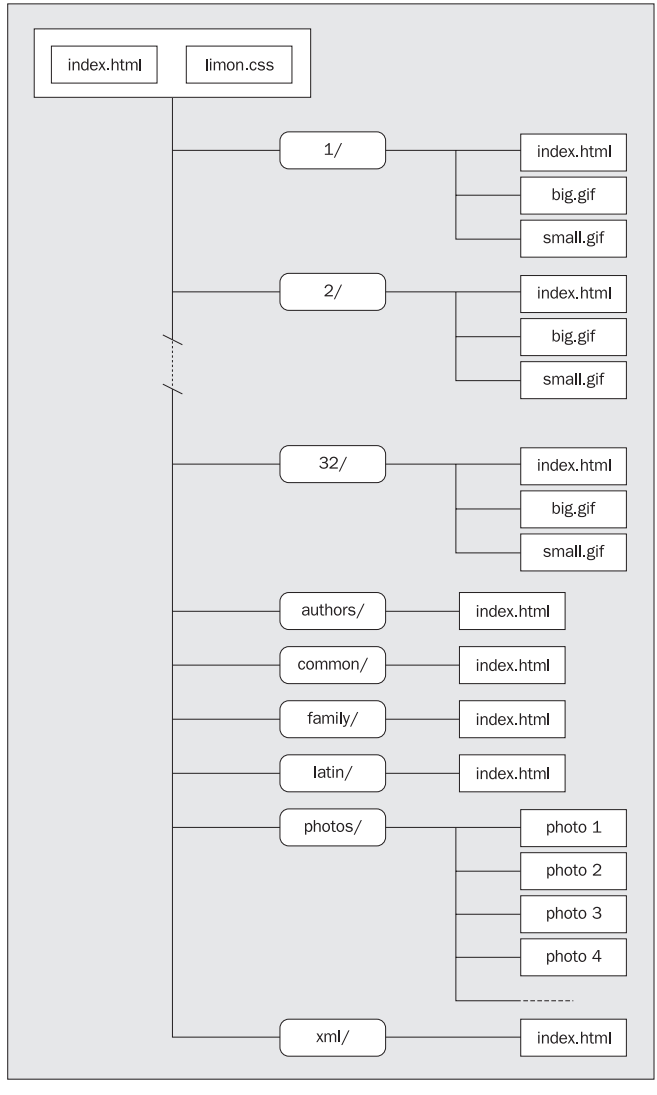


Рис. 12.3. Новая структура сайта

Вид страниц

Было решено, что прототипом стиля моего узла будет служить Web-страница для одного растения. Наверху располагается цветная строка, дающая номер и латинское название растения, а внизу аналогично выглядящая цветная строка с навигационными ссылками. Небольшая иллюстрация находится в правой стороне страницы; это изображение является гиперссылкой на более масштабное изображение растения. Слева располагается его описание вместе с обозначением семейства, латинским названием и списком народных названий.

```
<IMG SRC="screen/plant6.gif">
```

В следующем разделе приведен HTML для растения номер 6. Головные тэги `<HEAD>` документа заполнены тэгами `<META>`, которые должны рассказывать поисковым машинам о документе. Тэг `<LINK>` используется для того, чтобы указывать на поисковые серверы, на агентов пользователя, а также чтобы сообщить браузерам, где можно найти таблицы стилей для этого узла. При помощи атрибута `"LANG"` в тэге `<HTML>` мы сообщаем поисковым машинам, переводчикам и браузерам, что языком по умолчанию для нашего документа является английский.

Страница - 6/index.html

```
<HTML LANG="en">
<HEAD><TITLE>6 Momordica charantia L.</TITLE>
<META NAME="ROBOTS" CONTENT="ALL" LANG="en">
<META NAME="Author" CONTENT="Olivia S. Direnzo and Paul A. Houle" LANG="en">
<META NAME="Date" CONTENT="14 Jun 1998 18:31:55 GMT" LANG="en">
<META NAME="Copyright" CONTENT="&copy; 1998 Honeylocust Media Systems
(http://www.honeylocust.com/)" LANG="en">
<META NAME="Keywords" CONTENT="El Limon,Weeds,Botany,xml,Cucurbitacea,
Momordica charantia L.,balsam pear,balsam apple,cerasee bush,archucha,
balsamina,achochilla,pepinillo,cunde amor,melao de Sao Caetano,carcilla"
LANG="en">
<META NAME="Description" CONTENT="A collection of descriptions and illustrations
of weeds observed in El Limon, a small village in the Dominican Republic during
January 1998. " LANG="en">
<LINK REL="Index" HREF="../common/index.html">
<LINK REL="Index" HREF="../family/index.html">
<LINK REL="Index" HREF="../latin/index.html">
<LINK REL="Begin" HREF="../1/index.html">
<LINK REL="Top" HREF="..">
<LINK REL="Contents" HREF="..">
<LINK REL="Start" HREF="..">
<LINK REL="Prev" HREF="../5/index.html">
<LINK REL="Next" HREF="../7/index.html">
<LINK REL="STYLESHEET" HREF="../limon.css" TYPE="text/css">
</HEAD>
```

Для начала создаем в теле документа строку заголовков наверху страницы. Мы используем каскадные таблицы стилей (CSS), так что атрибут `CLASS` в `<TD CLASS=navbar>` устанавливает цвет фона в ячейке таблицы, а также цвет и размер текста в ячейке. Впоследствии обсудим этот вопрос подробнее.

```
<BODY BGCOLOR=#FFFFFF>
<TABLE CELLSPACING=0 WIDTH=100%>
<TR><TD CLASS=navbar>6 Momordica charantia L.</TD></TR>
</TABLE>
<A HREF="big.gif">
<IMG
SRC="small.gif" HEIGHT=396 WIDTH=278 ALIGN=RIGHT ALT="[Momordica charantia L.
illustration]" BORDER=0>
</A>
```

Далее идет текст описания растений. Здесь мы используем тэг <SPAN> HTML 4.0, чтобы пометить язык, на котором даются названия растений. Сам по себе тэг не оказывает воздействия на то, как браузер представляет текст, но позволяет установить границы раздела текста, над которым будут работать таблицы стилей.

```
<B>Family Cucurbitacea</B>
<BR>Latin name: <I><SPAN LANG="la">Momordica charantia L.</SPAN></I>
<BR>Common names: <I> balsam pear, balsam appl, cerasee bush,
<SPAN LANG="es">archucha</SPAN>, <SPAN LANG="es">balsamina</SPAN>, <SPAN
LANG="es">achochilla</SPAN>, <SPAN LANG="es">pepinillo</SPN>, <SPAN
LANG="es">cunde amor</SPAN>, <SPAN LANG="es">melao de Sao Caetano</SPAN>, <SPAN
LANG="es">carcilla</SPAN></I>
<P>

   Vine, climbs by tendrils. Leaves are alternate, soft and lightly hairy. Leaves
are deeply lobed with five lobes. (Length about 3 cm ) Yellow flowers arise from
pistils and stamen at center. (Diameter about 1.5 cm ) Pods are oval tapering to
a point with rows of little spikes, green turning orange as they mature. Exploded
pods show bright orange peels and four red seeds. Inside is sticky. Pod length
(about 2.5 cm ) Stem is hairy, very hairy at terminal end. Found growing on fence
along main road in full sun.

<P><B>Additional resources:</B>

<UL>

<LI><A HREF="big.gif">Large botanical illustration</A>

</UL>
```

Наконец мы создаем навигационную строку и добавляем уведомление об авторских правах. Я обнаружил, что для управления цветом *привязок* (anchors) приходится вставлять стиль в сами тэги привязки.

```
<BR CLEAR=ALL>
<TABLE CELLSPACING=0 WIDTH=100><TR><TD CLASS=navbar>
<A CLASS=navlink HREF="../5/index.html">PREVIOUS</A>
<A CLASS=navlink HREF="../index.html">TOP</A>
<A CLASS=navlink HREF="../7/index.html">NEXT</A>
<A CLASS=navlink HREF="../family/index.html">FAMILY</A>
<A CLASS=navlink HREF="../latin/index.html">LATIN</A>
<A CLASS=navlink HREF="../common/index.html">COMMON</A>
</TD></TR></TABLE>
<BR CLEAR=all><HR>
```

```
<I>Version 0.94a &copy; 1998 <A HREF="http://www.honeylocust.com/">Honeylocust
Media Systems.</A>
   Contact: <A HREF="mailto:houle@msc.cornell.edu">houle@msc.cornell.edu</A>
</BODY></HTML>
```

Таблицы стилей

В первой, черновой версии «Сорняков...» каскадные таблицы стилей не использовались, в результате оказалось, что этот вариант был в значительной степени несовместим со старыми браузерами, в частности с Netscape 2. Проблема заключалась в том, что Netscape 2 распознает тэг `<FONT COLOR>`, но ничего не знает об атрибуте `<TD BGCOLOR>`, используемом для установки фонового цвета ячейки таблицы. В результате навигационная строка и строка заголовков получаются невидимыми: белый текст на белом фоне.

```
<IMG SRC="screen/plant6ns2old.gi">
<CAPTION>Plant 6, before stylesheets, on Netscape 2</CAPTION>
```

Первая версия была реализована после того, как мы проверили «Сорняки...» с Netscape 4 на Unix и Windows и с IE 4 на Windows и Mac. Первая группа людей, посетивших узел, работала с передовыми Web-технологиями, около 80% посетителей имели браузер 4.0. Мы обнаружили проблему с `<FONT COLOR>`, когда показывали страницу человеку, использующему Netscape 2.

Совместное существование нескольких платформ, браузеров и версий браузеров – это неизбежная реальность жизни в Web, и в ближайшее время, когда люди будут выходить на связь с помощью телевизоров, сотовых телефонов и даже Web-браузеров, установленных в киосках на углах улиц, положение станет еще хуже. Можно, конечно, ограничиться одной версией, одним браузером или одной платформой, но это оттолкнет многих пользователей, а в будущем, когда сегодняшняя наиновейшая версия станет пережитком прошлого, подобное решение будет выглядеть особенно нелепо.

Очень важно овладеть стратегией совместимости, прочувствовать ее. Первое решение, к которому я пришел, заключалось в том, что мне необходимо поддерживать одну и только одну версию «Сорняков...». При этом сразу пришлось столкнуться с одним из наиболее часто задаваемых вопросов о CGI-скриптах: «Как мне снабжать разными вариантами страниц пользователей, имеющих различные браузеры?» Я ответил на него следующим образом – а зачем вообще этим заниматься? Стоит ли поддерживать сразу нескольких версий страниц? Эта работа в конце концов превратится в сплошной кошмар. Вам не просто придется поддерживать разные версии. Сам по себе код, детектирующий тип браузера, станет местом, где система, скорее всего, будет давать сбой. Поскольку все большее число людей выходят на Web при помощи proxy-серверов, становится все труднее и труднее выяснить, какой именно браузер используется. Провести черту, четко определяющую, какие браузеры поддерживать, а какие нет, очень сложно, поскольку при этом сотни более или менее распространенных браузеров останутся за бортом. А что произойдет, когда появятся пятые версии браузеров? Как только я захочу специализировать свой продукт в другом ключе, сразу даст знать о себе эффект умножения. Например, если я поставлю цель разработать версии для трех различных браузеров на двух различных языках, да еще с возможностью выбора качества

изображения (одного из двух) – это приведет к слишком большому количеству комбинаций, которые надо поддерживать.

Вот почему я отдал предпочтение единственной версии. Это будет узел для версии 4.0 основных браузеров (Netscape и Microsoft) с требованием, чтобы и остальные браузеры могли в нем разобраться. В такой ситуации идеальное решение обеспечивают таблицы стилей. Вместо того, чтобы использовать тэг `<FONT>`, который может привести к беспорядку, когда браузер понимает его частично, было принято решение позволить тексту выглядеть просто текстом в старых браузерах, а для новых версий снабдить его таблицами стилей. Хотя браузеры 4.0 поддерживают *CSS-позиционирование* (CSS-Positioning), которое в принципе позволяет таблицам стилей установливать точный контроль над местом расположения текста, я все же решил для форматирования использовать таблицы, потому что, с одной стороны, оба основных браузера реализуют свои собственные подмножества стандарта CSS-P, в с другой – создание таких таблиц не вызывает особых проблем.

Поскольку таблицы стилей находятся в стадии развития, будет правильно протестировать страницы таблиц стилей на обоих основных браузерах. Существует множество мелких загвоздок в их реализации, например, выяснилось, что браузер Netscape не может выставить свойство текста `font-size` до тех пор, пока не указано также свойство `font-family`. Но и в этих условиях задача, заключающаяся в том, чтобы заставить CSS согласованно изменять `<FONT COLOR>` и фон таблицы совместимо как для Netscape, так и для IE 4, – окажется несложной, хотя браузеры действительно различаются тем, как они поддерживают CSS-позиционирование.

Совет *Превосходный учебный материал и ссылки на таблицы стилей можно найти в книге Professional Stylesheets for HTML and XML («Профессиональные таблицы стилей для HTML и XML»), написанной Фрэнком Бумфреем (Frank Boumphrey) и опубликованной в 1998 издательством Wrox Press, ISBN 1-961001-65-7.*

Поскольку в создании таблиц стилей я был новичком, мне пришло в голову начать с адаптирования заглавной страницы к таблицам стилей. Дело в том, что эта страница намеренно выполняется в точном соответствии с остальными. Добившись поставленной цели, я мог заменить компилятор моего узла, чтобы начать записывать HTML-код для таблиц стилей. Таблицы стилей для «Сорняков...» – это `limon.css`.

Таблица стилей limon.css

```
/* Стили, используемые на сайте. */

.navbar { background-color:green;
        color:white;
        font-family:sans-serif;
        font-size:large; }

.navlink { color:white;
        text-decoration:none; }

.unknown{  color:red;
        font-weight:bold; }

BODY { background-color:white;
```

```
        font-family:times,serif; }

/* Стили, используемые на заглавной странице. */

.menubox { background-color:green;
        background-image:url(leaves.jpg);
        color:white;
        font-family:times,serif;
        font-size:large;}

.menulink ( color:white; }

/* Стили, используемые для дополнения xml. */

.codebar { background-color:blue;
        color:white; }

.codebody { background-color:yellow;
        color:black; }

/* Стили для раздела "Об авторах". */

DIV.head { font-family:helvetica,sans-serif; font-size: 48px ; color: #f800f8;}
DIV.titles {font-family:helvetica,sans-serif; font-size: 30px ; color: teal;}
#oliviahead {position: absolute ; top: 50px ; left: 130px;}
#oliviatitles {position: absolute ; top: 100px ; left: 150px;}
#paulhead {position: relative; top: -100px ; left: -130px;}
#paultitles {position: relative; top: -100px ; left: -100px;}
#blurb {position: relative; top: -78px;}
```

Все страницы, составляющие узел, имеют одну и ту же таблицу стилей. Они ссылаются на нее, поскольку содержат тэг

```
<LINK REL="STYLESHEET" HREF="../limon.css" TYPE="text/css">
```

в тэге <HEAD> документа. В виду того, что вся информация о стиле теперь находится в одном месте, у меня появляется возможность, редактируя таблицу стилей, играть фоном, цветами ссылок и аналогичными характеристиками всего узла. Принятое решение заключалось в том, чтобы непрерывно адаптировать программу к использованию таблиц стилей. Я подумывал добавить переключатель на код, выдающий HTML-документ в старом стиле, но решил, что эта операция не стоит труда.

Версия документа с таблицами стилей не только лучше совместима с многочисленными браузерами, она еще и проще. В старой версии код HTML для строки заголовков оформлялся так:

```
<TD BGCOLOR=#000000>
   <FONT COLOR=#FFFFFF>
   <FONT SIZE=+2>
   <FONT FACE='sans-serif'>6 Motordica charantia L.</FONT>
   </FONT>
   </FONT>
</TD>
```

а в новой версии эта запись выглядит следующим образом:

```
<TD CLASS=navbar>6 Momordica charantia L.</TD>
```

Усовершенствование существенное! Другое преимущество таблиц стилей заключается в том, что вид узла может быть изменен без необходимости перекомпиляции. Однажды я решил, что строка заголовков и навигационная строка должны быть зелеными, а не черными, и быстро осуществил изменения, подредактировав таблицу стилей.

Вот как новая страница теперь выглядит в Netscape 2. Она смотрится немного непритязательно, но ничего не пропущено.

```
<IMG SRC="csreen/plant6ns2.gif">
<CAPTION>Plant 6 in the style-sheet version seen with Netscape 2</CAPTION>

<H4>Index pages</H4>

<IMG SRC="screen/latin.gif">
<CAPTION>index by latin name</CAPTION>

<IMG SRC="screen/common.gif">
<CAPTION>index by latin name</CAPTION>
```

Выбрав способ отображения индивидуальных страниц с растениями, мне также предстояло спроектировать страницу указателя. Строка заголовков и навигационная строка будут, естественно, такими же. Поскольку расположение указателя в одну колонку приводит к большой потере экранного пространства, я поместил указатель в две колонки, используя таблицы. HTML-код для таблицы выглядит следующим образом:

```
<TABLE WIDTH=100% COLS=2>
   <TR>
   <TD WIDTH=50% VALIGN=top>
   ..left hand column..
   </TD>
   <TD WIDTH=50% VALIGN=top>
   ..right hand column..
   </TD>
   </TR>
</TABLE>
```

Если указаны ширина всех ячеек таблицы и число столбцов, Netscape 4.0 и Internet Explorer 4.0 могут правильно вывести таблицу уже тогда, когда загрузилась только некоторая ее часть. Указатели названий (и латинских, и народных) созданы по существу одним и тем же кодом – Java-классами, которые, оформляя указатели, унаследовали большинство своих возможностей от обычных классов. В код, создающий указатель, я вставил опцию, добавляющую крупные заглавные буквы в начало каждого нового раздела, начинающегося со следующей буквы алфавита. Поскольку у нас много народных названий, я сделал эту опцию разрешенной и для народных названий, но запретил для латинских, поскольку крупные заглавные буквы выглядят нелепо, когда под ними находится всего одно наименование.

```
<IMG SRC="screen/family.gif">
```

Указатель по семействам аналогичен двум другим индексам, но организован при помощи HTML, что позволяет сгруппировать вместе растения, относящиеся к одному семейству.

Метаинформация

Таким образом я воспользовался преимуществами новых возможностей в HTML 4.0, которые позволяют добавлять в документ метаинформацию, и на этой основе принял несколько конструкторских решений. В результате «за кадром» остались ненужные пользователю операции. Эта методика детально описана в официальной спецификации, расположенной на сайте

http://www.w3.org/TR/REC-html40/

Одна из этих важных характеристик заключается в том, что теперь можно указать язык всего документа или его частей, если документ составлен из фрагментов, написанных на разных языках. Другими словами, в случае обозначения языка программа-переводчик способна избежать ошибок, возникающих при попытке перевести текст, который не следует переводить. Это важно для «Сорняков Эль Лимона», поскольку в него включены латинские наименования, кроме того, там помещены народные названия на английском и испанском языках. Тэг

```
<HTML LANG="en">
```

например, сообщает любому браузеру, поисковой машине или другой программе, что языком по умолчанию для этой страницы является английский. Любая часть текста может быть отмечена тэгом `SPAN`, как написанная на другом языке, а именно:

```
<SPAN LANG="es">Loco Cabayo</SPAN>
```

Этот фрагмент отмечен как написанный на испанском языке. Многие другие тэги также содержат атрибут `LANG`, например:

```
<META NAME-"keywords" LANG="en" CONTENT="Weeds,Botany,XML">
```

HTML 4.0 также добавляет новые возможности, позволяющие «рассказать» браузерам и поисковым машинам о структуре вашего документа. Эта перспектива очень интересна, потому что я, как и любой пользователь, бываю сильно разочарован, когда поисковая машина выдает мне URL страницы, являющейся частью большой коллекции, а я не могу найти заглавную страницу узла. Это неудобство можно предотвратить добавлением тэга `<LINK>` в элемент `<HEAD>` документа:

```
<LINK REL=TOP HREF="http://www.honeylocus.com/limon/">
```

Другие типы связей документа, такие как `INDEX`, `APPENDIX`, `NEXT` и `PREV`, могут быть указаны в атрибуте `REL`. В добавление к этим новым возможностям, существует много тэгов `<META>`, которые распознаются внутри тэга `<HEAD>` документа большей частью поисковых машин, например:

```
<META NAME=AUTHOR CONTENT="Paul Houle (paul@honeylocust.com)">
<META NAME=DATE CONTENT="31 May 1998>
<META NAME=COPYRIGHT CONTENT="&copy; 1998 Honeylocust Media Systems">
<META NAME=KEYWORDS CONTENT="El Limon,Weeds, Botany,XML">
<META NAME=DESCRIPTION CONTENT="A collection of descriptions and
illustrations...">
```

В описании кода, создающего HTML-документ, а именно `GenerateHtml.java`, вы найдете строки, с помощью которых добавляется метаинформация.

Заглавная страница и дополнения

Понятно, что для «Сорняков Эль Лимона» необходима *заглавная* (top) страница. Чтобы придать сайту согласованный вид, ее следует разработать таким образом, чтобы она была похожа на все остальные страницы. Скорее всего, мне удалось бы создать ее автоматически, что облегчило бы согласование ее вида со всем текстом, однако куда проще написать материал вручную, используя каскадные таблицы стилей. Очевидно, что для этого мне было необходимо сохранить строку заголовков, но я решил опустить навигационную строку, поскольку заглавная страница и так уже имеет навигационные ссылки. Вместо нее были включены простое меню и небольшие участки текста с гиперссылками. Размещение изображения в ячейке таблицы – это самый простой способ украсить страницу.

```
<IMG SRC="screen/top.gif">
<CAPTION>Top page</CAPTION>
```

В настоящее время программа «Сорняки...» имеет два дополнения, которые также были созданы вручную. Оба используют `limon.css`, в качестве таблицы стилей. Одно дополнение – это краткое техническое эссе о странице (очень короткая версия данного описания), а другое – реклама авторов, в которой использовано CSS-позиционирование, чтобы поместить слова прямо на изображение и добиться красочных эффектов в браузерах 4.0. При написании этой страницы выяснилось, что браузеры 4.0 плохо совмещаются друг с другом, более того, и со спецификацией консорциума W3 в области CSS-позиционирования. Ни один из браузеров не исполняет правый и нижний атрибуты, которые позволили бы легко позиционировать элементы по отношению к правому краю страницы. В результате мне пришлось сражаться с многочисленными вывертами обоих браузеров, как Netscape, так и IE, в которых я до сих пор не до конца разобрался. К сожалению, в старых браузерах страница все еще выглядит не очень привлекательно. Однако рекламная полоса не имеет особого значения, так что это место более подходит для показа трюков, которые пока еще нельзя выносить на участки серьезного текста.

Скрытая, но сложная проблема

Была еще одна проблема, сумевшая прокрасться в HTML-конструкцию «Сорняков...». Созданная издательская система позволила легко ее исправить, что доставило мне, автору, огромное удовольствие. Однажды вечером мы с Оливией отправились в общественную компьютерную лабораторию в местном колледже, где на множестве старых компьютеров Mac был установлен Netscape 1.1. Просматривая наши страницы, мы обнаружили, что Netscape 1.1 не понимает одиночные кавычки в `<A HREF='url'>`, так что все наши тэги ссылок и изображений вышли из строя. Как оказалось, это обстоятельство также препятствует работе инструмента `pavuk` (средству *Web-отображения* (mirroring), который нам очень нравится) переходить по многим из ссылок. Кстати, его адрес:

http://www.idata.sk/%7Eondrej/pavuk/

Поскольку более новые браузеры позволяют пользоваться как одиночными, так и двойными кавычками, мы даже не подозревали, что одиночные кавычки не работают со старыми браузерами. Безусловно, Netscape 1.1 – очень старый браузер,

но применение одиночных кавычек является слишком нелепой причиной, чтобы перекрывать доступ пользователям к продукту, который создавался специально для работы в странах третьего мира, где люди не имеют новейшего программного обеспечения и новейших компьютеров. Поскольку все наши страницы создаются программой, нам было легко решить эту проблему, просто исправив код и перекомпилировав страницы. Если бы я писал HTML-страницы вручную, я мог бы потратить уйму времени, выискивая каждое появление одиночных кавычек. Мой скрипт, осуществляющий глобальный поиск и замену, безусловно, помог бы, но процесс явно не доставил бы мне удовольствия.

Отметим, что мы были вынуждены отыскивать решения этих проблем самостоятельно. Хотя протокольные записи нашего сервера показывали, что некоторые посетители просматривали наши страницы при помощи Netscape 2 и Netscape 1.1, пользователи Web известны тем, что не сообщают о проблемах, возникающих при просмотре ваших страниц. Однажды я написал с помощью JavaScript страничку, которая вывела из строя сотни старых браузеров прежде, чем я встретил лично того, кто сообщил мне об этом случае. Если вы хотите найти неполадки в ваших Web-страницах, вам обязательно надо протестировать их самостоятельно.

Заключение

Дизайн нашего HTML-документа был серьезно ограничен тем, что мы хотели создать уникальный внешний вид, не используя текст в виде изображений, так как некоторые клиенты могли воспользоваться программами автоматического перевода. Со временем мы изменили свой HTML-дизайн, поскольку обнаружили ряд несовместимостей со старыми браузерами и проблемы при просмотре страниц из локальной файловой системы. Мы остановились на использовании каскадных таблиц стилей как на способе установки стиля. Эта методика создает минимум проблем для старых браузеров, при этом было обнаружено, что она существенно упрощает HTML-код.

Генерирование HTML-кода

Класс Generator.java

Класс `Generator.java` – это стержень нашей программы, создающей HTML-код. `Generator.java` использует объекты с уровня ввода и уровня данных, чтобы прочитать содержимое XML-файлов в память, а затем применяет объекты с уровней вывода, чтобы создать Web-страницы. `Generator.java` имеет простой интерфейс командной строки, поскольку программа не настолько сложна, чтобы обладать графическим интерфейсом. Если вы все равно захотите написать GUI, то этот класс вам придется модифицировать.

Начнем с основ:

```
package honeylocust.limon;

import honeylocust.limon.representation.ImageAttach;
import honeylocust.limon.representation.MSXMLSpeciesFactory;
import honeylocust.limon.representation.Species;

import java.io.File;
```

```
import java.util.Vector;
import java.util.Enumeration;
```

Единственным методом в «генераторе» является метод `static main`, почти как в программах на C, начинающихся с `main()`. Сейчас самое время упоминуть о политике по отношению к исключениям, которой я стараюсь придерживаться. Поскольку язык Java очень сильно ограничивает использование исключений, вам, приступая к программированию на этом языке, сразу придется вступить в отчаянную схватку с компилятором. Мне потребовался примерно год, чтобы добиться с ним согласия. Вашим первым порывом будет перехватывать исключения вблизи тех мест, где они появляются. Это неправильно, поскольку обработка исключений в Java – это хороший способ организовать глобальную обработку ошибок. Для работы с такой системой важно осознать, что перехватывать исключение следует только в том случае, если вам это нужно. В результате я написал метод `main()` так, что он перехватывает любые исключения, появляющиеся ниже него, и печатает трассировку стека. (Это и является целью `try` и `catch`).

Первая часть этой программы ответственна за преобразование XML-файлов в объекты `Species`. В командной строке содержится список имен файлов, каждый из которых является XML-документом, содержащим описания трав. Программа выполняет цикл по аргументам командной строки и анализирует каждый файл. Объект `ImageAttach` используется для нахождения местоположения файлов с изображениями, относящимися к каждому растению. Затем создается объект `PlantIndex`, который необходим для поддержания простой базы данных о растениях, позволяющей искать растения по названиям.

После того как описания растений подготовлены и размещены в нашей простой базе данных, программа готова создавать Web-страницы для каждого растения, так же как и указатели по латинским и народным названиям и по названиям семейств.

```
public class Generator {

    public static void main(String argv[]) {
    try {
        Vector v=new Vector();
        for(int i=0;i<argv.length;i++)
            MSXMLSpeciesFactory.parseLimon(v,new File(argv[i]));

        ImageAttach ia=new ImageAttach(new File("32colors"));
        ia.attachImages(v);

        PlantIndex pi=new PlantIndex();
        pi.addPlants(v);

        File basePath=new File("output");
        GenerateHtml.setHtmlPath(basePath);
        GenerateHtml.setCSSPath("../limon.css");

        for(Enumeration e=v.elements();e.hasMoreElements();) {
            Species s=(Species) e.nextElement();
            System.out.println("Generating"+s.getId());
            (new GeneratePlant(pi,s)).generate();
        };
```

```
            GenerateLatin glatin=new GenerateLatin(pi);
            glatin.generate();

            GenerateCommon gcommon=new GenerateCommon(pi);
            gcommon.setAddLetters(true);
            gcommon.generate();

            GenerateFamily gfamily=new GenerateFamily(pi);
            gfamily.generate();

        } catch(Exception e) {
            e.printStackTrace();
        }

        System.exit(0);
    }

    }
```

Следующие четыре класса относятся к обслуживающим, то есть предоставляющим функции, которыми будут пользоваться другие классы. Класс `PlantIndex` является простой базой данных, которая позволяет отыскивать растения по названиям, а класс `MultiHashtable` используется классом `PlantIndex` для связывания одного названия семейства с более чем одним растением, принадлежащим этому семейству. Класс `WebImage` автоматизирует процесс создания тэга `<IMG>`, предоставляя интерфейс для сложного синтаксиса тэга `<IMG>`, а также автоматически отыскивая ширину и высоту изображения и внося эти величины в тэг. При нахождении размеров изображения класс `WebImage` зависит от класса `GifInfo`.

Класс PlantIndex.java

Со временем мне стало понятно, что для облегчения создания HTML-указателей растений, мне не обойтись без простой базы данных, которая позволит отыскивать растения по их латинскому и народному названиям, по названию семейства, а также по номеру ID. Поскольку все эти действия образуют четко определенный комплекс функциональных возможностей, я решил поместить все функции, связанные с индексированием, в один класс `PlantIndex.java`. Чтобы связать ключи – номер ID, латинское и народное название – с растениями, класс `PlantIndex` использует `java.util.Hashtable`. Поскольку к одному семейству обычно принадлежит более чем одно растение, а хэш-таблица может связать с ключом только одно значение, мне пришлось разработать свой собственный класс `MultiHashtable` (который описывается после `PlantIndex`), позволяющий ключу указывать более чем на одно растение. Начинаем с инициализации хэш-таблиц:

```
package honeylocust.limon;

import honeylocust.limon.representation.LanguageString;
import honeylocust.limon.representation.Species;

import java.util.Vector;
import java.util.Enumeration:
import java.util.Hashtable;

public class PlantIndex {
```

```
    Integer d_maxPlantNumber;
    Integer d_minPlantNumber;

    Hashtable d_plantsById;
    Hashtable d_plantsByLatin;
    Hashtable d_plantsByCommon;
    MultiHashtable d_plantsByFamily;

    Hashtable d_languageCommon;

    public PlantIndex() {
    d_minPlantNumber=null;
    d_maxPlantNumber=null;

    d_plantsById=new Hashtable();
    d_plantsByLatin=new Hashtable();
    d_plantsByCommon=new Hashtable();
    d_plantsByFamily=new MultiHashtable();
    };
```

Следующим шагом после инициализации `PlantIndex` является заполнение хэш-таблиц описаниями растений и их ключами.

```
public synchronized void addPlants(Vector v) {
for(Enumeration e=v.elements();e.hasMoreElements();)
    addPlant((Species) e.nextElement());
};

public synchronized void addPlant(Species s) {
d_plantsById.put(s.getId(),s);

try {
    int plantNumber=Integer.parseInt(s.getId());
    if (d_maxPlantNumber==null || d_maxPlantNumber.intValue() < plantNumber)
        d_maxPlantNumber=new Integer(plantNumber);

    if (d_minPlantNumber==null || d_minPlantNumber.intValue() < plantNumber)
        d_minPlantNumber=new Integer(plantNumber);

} catch(NumberFormaException e) {
};

if (s.getFamily()!=null)
    d_plantsByFamily.put(s.getFamily(),s);
else
    d_plantsByFamily.put(new LanguageString("Unknown").intern(),s);

if (s.identified()) {
    LanguageString latin[]=s.getLatin();
    for(int i=0;i<latin.length;i++) {
    LanguageString lname=latin[i];
        if (d_plantsByLatin.get(lname)!=null)
            System.err.println("Warning:duplicate latin name->
                    "+lname.toString());

        d_plantsByLatin.put(lname,s);
    };
```

```
    LanguageString common[]=s.getCommon();
    for(int i=0;i<common.length;i++) {
    LanguageString cname=common[i];
        if (d_plantsByCommon.get(cname)!=null)
            System.err.println("Warning:duplicate common name -> "+cname);
        d_plantsByCommon.put(cname,s);
    };
};
};
```

Затем предоставляем несколько функций для доступа к индексам:

```
public Species getById(String key) {
return (Species) d_plantsById.get(key);
};

public synchronized LanguageString[] getLatinNames() {
return getKeyArray(d_plantsByLatin);
};

Language String[] getKeyArray(Hashtable h) {
LanguageString s[]=new LanguageString[h.size()];
int i=0;
for(Enumeration e=h.keys();e.hasMoreElements();)
   s[i++]=(LanguageString) e.nextElement();

return s;
};

public Species getByLatin(LanguageString key) {
return (Species) d_plantsByLatin.get(key);
};

public synchronized LanguageString[] getCommonNames() {
return getKeyArray(d_plantsByCommon);
};

public Species getByCommon(LanguageString key) {
return (Species) d_plantsByCommon.get(key);
};

public synchronized LanguageString[] getFamilies() {
return (LanguageString[])d_plantsByFamily.keyArray();
};

public Species getByFamily(LanguageString key) {
return (Species) d_plantsByFamily.get(key);
};

public LanguageString speciesName (Species s) {
if (!s.identified())
   return new LanguageString("Plant "+s.getId());

return((s.getLatin())[0];
};
```

и несколько функций для доступа к id-номерам растений:

```
    public int getMaxPlantNumber() {
    return d_maxPlantNumber.intValue();
    };

    public int getMinPlantNumber() {
    return d_minPlantNumber.intValue();
    };

    public boolean isNumericId() {
    return (d_maxPlantNumber!=null);
    };

    public Integer plantNumber(Species s) {

    boolean numeric=true;
    int myNumber=0;

try {
        myNumber=Integer.parseInt(s.getId());
    } catch(NumberFormatException e) {
        numeric=false;
    };

    if (!numeric)
        return null;

    return new integer(myNumber);
    };
};
```

Класс MultiHashtable.java

Класс `MultiHashtable` очень прост, он весьма похож на `Hashtable` за исключением того, что в нем каждый ключ может иметь несколько значений. Вместо того чтобы изобретать свои собственные структуры данных (это могло оказаться хорошей идеей, если бы требовалось, сделать их очень быстрыми или очень хорошо масштабируемыми), я решил сократить свою работу, используя хэш-таблицу. Она устанавливает соответствия ключей с векторами `java.util.Vectors`, каждый из которых является расширяемым массивом, отслеживающим множественные значения.

```
package honeylocust.limon;

import honeylocust.limon.representation.LanguageString;
import honeylocust.limon.representation.Species;

import java.util.Vector;
import java.util.Hashtable;
import java.util.Enumeration;

public class MultiHashtable {

    Hashtable p_ht;

    public MultiHashtable() {
    p_ht=new Hashtable();
```

```
    };
    public synchronized void put(Object key,Object value) {
    Vector v=(Vector) p_ht.get(key);
    if (v==null) {
        v=new Vector();
        p_ht.put(key,v);
    };
    if(!v.contains(value)) {
        v.addElement(value);
    };
    };
    public synchronized Species[] get(Object key) {
    Vector v=(Vector) p_ht.get(key);
    if (v==null)
        return new Species[0];
    Species o[]=new Species[v.size()];
    v.copyInto(o);
    return o;
    };
    public Enumeration keys() {
    return p_ht.keys();
    };
    public LanguageString[] keyArray() {
    LanguageString o[]=new LanguageString[p_ht.size()];
    int i=0;
    for(Enumeration e=keys();e.hasMoreElements();)
        o[i++]=(LanguageString) e.nextElement();
    return o;
    };
};
```

Объект WebImage.java и класс GifInfo.java

HTML-тэг `<IMG>` является сложным тэгом, поскольку имеет много атрибутов, таких как `SCR`, `ALT` и `ALIGN`. Особенно докучает необходимость устанавливать атрибуты `HEIGHT` и `WIDTH` для каждого изображения. Чтобы облегчить эту задачу, я создал объект `WebImage`, который представляет собой одиночный тэг `<IMG>`, позволяющий управлять его атрибутами посредством методов и таким образом автоматически устанавливать размеры изображения. При определении размера изображений `WebImage` зависит от другого класса, `GifInfo.java`, который будет описан позднее. Если указать `WebImage` путь к месту размещения изображения, а также другие атрибуты, он автоматически создает тэг `<IMG>`. Первая часть `WebImage` отвечает за сбор информации об изображении, а вторая вставляет изображение на страницу:

```
/* Сбор информации об изображении. */
```

```
package honeylocust.limon;

import java.io.PrintWriter;
import java.io.File;
import java.io.FileNotFoundException;

public class WebImage {

   static GifInfo d_gifinfo=new GifInfo();

   public String d_src;
   public String d_alt;
   public String d_align;
   public String d_webpath;
   public boolean d_border;

   int d_width,d_height;

   public WebImage(File dir,String src) throws
InterruptedException,FileNotFoundException {
   d_src=src;
   File f=new File(dir,src);

   if (!(f.exists()))
       throw new FileNotFoundException(f.getAbsolutePath());

   int z[]=d_gifinfo.getDimensions(f);
   d_width=z[0];
   d_height=z[1];
   d_webpath="";
   d_border=true;
   };

   public void setWebpath(String s) {
   d_webpath=s;
   };

   public void setAlign(String s) {
   d_align=s;
   };

   public void setAlt(String s) {
   d_alt=s;
   };

   public void setBorder(boolean b) {
   d_border=b;
   };
...
...
/* Вставка изображения на страницу */
   public void insert(PrintWriter w) {
   w.print("<IMG SRC=\""+d_webpath+d_src+"\" HEIGHT="+d_height+
           "WIDTH="+d_width);
   if(d_align!=null)
       w.print(" ALIGN="+d_align);
```

```
        if(d_align!=null)
            w.print("ALT=\""+d_alt+"\"");

        if(!d_border)
            w.print("BORDER=0");

        w.println(">");
        };

    };
```

Класс `GifInfo.java` является обслуживающим классом, облегчающим определение размеров изображения в неграфической программе. В Java AWT имеются очень мощные возможности для работы с изображениями, но они дают больше, чем требуется. На самом деле мне бы хотелось вызвать такой метод, который бы принял название файла и просто вернул размер изображения. AWT не способен обеспечить подобную операцию. Вместо этого он предоставляет систему загрузки изображений через Web. Можно начать процесс, вызвав метод `prepareImage()` любого компонента AWT или набора инструментов `java.awt.Toolkit`, однако вызов метода возвращается раньше, чем закончится загрузка изображения. Такое поведение называется асинхронным. Для Web это очень хорошая возможность, поскольку загрузка изображения может занять много времени и будет неудобно, если программа вдруг окажется заблокированной во время загрузки. С другой стороны, моя программа выполняется из командной строки, и загрузка изображения с диска занимает всего секунду, так что мне не нужна вся мощь, предоставляемая этим методом. У класса `GetInfo` двоякая задача. Во-первых, он создает объект `java.awt.Toolkit`, который позволяет получить доступ к функциям AWT в неграфической программе, а во-вторых, он предоставляет синхронный метод для получения размера изображения. Иначе говоря, метод не возвращается до тех пор, пока не закончится его исполнение. Первая часть программы получает экземпляр `Toolkit` (инструментария) для доступа к функциям AWT:

```
package honeylocust.limon;

import java.awt.Panel;
import java.awt.Toolkit;
import java.awt.Image;
import java.awt.image.ImageObserver;
import java.io.File;

public class GifInfo;

    Toolkit d_tool;

    public GifInfo() {
    d_tool=(new Panel()).getToolkit();
    };
...
```

Метод `getDimensions()` проделывает основную работу. Он использует методы `getImage()` и `prepareImage()` объекта `Toolkit`, чтобы начать процесс загрузки изображения. Для уведомления о том, когда сведения об изображении становятся доступными, он создает экземпляр `GifInfoObserver`, который является внутренним

классом класса `GifInfo`. Для того чтобы преобразовать асинхронное действие `prepareImage` в синхронное поведение `getDimension()`, мне было необходимо понять систему обработки событий в Java. Коротко говоря, как только начинается загрузка изображения, метод `getDimensions()` применяет метод `wait()` к `GifInfoObserver`. Это приводит к приостановке процесса, который был инициализирован методом `getDimensions`. Когда AWT заканчивает загрузку изображения, один из каналов AWT вызывает метод `imageUpdate()` класса `GifInfoObserver`. Этот метод использует метод `notify()`, чтобы «пробудить» приостановленный поток, который только того и ждал. Затем метод `getDimensions()` может прочитать высоту и ширину изображения и вернуть информацию вызвавшему его методу.

```
...
public int[] getDimensions(File f) throws InterruptedException {
return getDimensions(f.getAbsolutePath());
};

public int[] getDimensions(String) throws InterruptedException {
Image i=d_tool.getImage(s);
int x[]=new int[2];

GifInfoObserver o=new gifInfoObserver(ImageObserver.ALLBITS);
synchronized(o) {
   d_tool.prepareImage(i,-1,-1,o);
   o.wait();
}

x[1]=i.getHeight(null);
x[0]=i.getWidth(null);

i.flush();
return x;
};

class GifInfoObserver implements ImageObserver {
int d_waitFor;

public GifInfoObserver(int waitFor) {
   d_waitFor=waitFor;
};

   public synchronized boolean imageUpdate(Image i,int flags,int x,int y,int
width,int height) {

   if ((flags&ImageObserver.ERROR)!=0) {
       System.out.println("Error in imageUpdate.");
       notify();
       return false;
   };

   if ((flags&d_waitFor)!=0)
       return true;
   notify();
   return false;
   };
```

```
    };
};
```

Класс GenerateHtml.java

При конструировании кода, создающего HTML, я столкнулся с любопытной задачей. В разрабатываемой программе должно было помещаться много похожих друг на друга Web-страниц, так что представлялось заманчивым использовать один и тот же код для создания навигационной строки, верхних и нижних колонтитулов и навигационных ссылок. Я хотел по возможности чаще пользоваться единым механизмом и при этом его конструкция не должна была отличаться особой сложностью. Решением оказался метод наследования.

Наследования в Java – дело серьезное, поскольку данный язык не поддерживает множественное наследование. Многие считают это существенным ограничением его возможностей, но, как утверждают разработчики Java, за счет того, что в нем имеется одиночное наследование, пользователи определенно остаются в выигрыше. Механизм наследования в Java намного проще, чем в C++, а это значит, что программистам его легче использовать и для него легче писать компиляторы. Те, кто только начинает пользоваться языком Java, склонны слишком часто обращаться к наследованию. Учтите, что в нашем запасе уловок это не единственный инструмент повторного использования кода. Кроме того, поскольку наследовать можно только из одного класса, такой способ весьма скуден на решения.

Итак, я решил создать абстрактный базовый класс для генерации кода «Сорняков...», называемый `GenerateHtml.java`, и построить иерархию, в которую нетрудно будет выборочно добавлять возможности для формирования различных типов страниц. Полосы для отдельных растений создаются классом `GeneratePlant`, который является подклассом класса `GenerateHtml`. Класс `GenerateIndex` содержит логические операции для генерации указателей по латинским и народным названиям. Сам по себе `GenerateIndex` является абстрактным классом, поскольку большая часть логических операций для полос с латинскими и народными названиями одинакова. Классы `GenerateCommon` и `GenerateLatin` – подклассы `GenerateIndex`, образованные простым добавлением методов, необходимых для получения списка ключей в указателе и для выявления ID-номера растения с данным ключом (для этой работы они вызывают `PlantIndex`). И наконец, класс `GenerateFamily` должен создавать существенно отличающийся тип указателя, поскольку одно и тоже название семейства может принадлежать более чем одному растению. Поэтому вместо того, чтобы непосредственно входить в иерархию класса `GenerateIndex`, этот класс является подклассом класса `GenerateHtml`.

Класс `GenerateHtml` предоставляет много возможностей, необходимых для создания Web-страниц. Он генерирует тэг `<HEAD>` страницы, тэги `<META>` и `<LINK>` для поисковых машин, а также строку заголовков наверху страницы и навигационную строку внизу. Кроме того, он предоставляет метод для вывода двухколоночного текста, который используется на страницах с указателями, метод для превращения `LanguageStrings` в HTML-код, размеченный информацией о языке, и несколько элементарных функций для работы с файлами, используемых для копирования файлов изображений из места их первоначального хранения в директории, где

размещаются Web-страницы. Далее показано, как выглядит первая часть кода.

```
package honeylocust.limon;

import honeylocust.limon.representation.LanguageString;
import java.io.PrintWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.util.Vector;
import java.io.util.Date;

public abstract class GenerateHtml {

   private static File s_htmlPath;
   private static String s_cssPath;

   PrintWriter d_out;
   File d_dir;
   GenerateHtml(File dir,String s) throws IOException {
   ensureDirectory(dir);
   d_out=new PrintWriter(new FileWriter(new File(dir,s)));
   d_dir=dir;
   };

   static File toHtmlPath(String s) {
   return new File(s_htmlPath,s);
   };

   static void setHtmlPath(File s) {
   s_htmlPath=s;
   };

   static void setCSSPath(String s) {
   s_cssPath=s;
   };
...
```

У этого класса нет конструктора, поскольку он не имеет полей данных, которые надо инициализировать. Имеются два параметра, `s_htmlPath` и `c_cssPath`, декларированные как статические, поскольку они совместно используются всеми экземплярами класса `GenerateHtml.java`. Таким образом, можно вызвать

```
GenerateHtml.setCSSPath(String s)
```

чтобы выставить местоположение совместно используемой каскадной таблицы стилей. Риск заключается в том, что программист может случайно испортить конфигурацию нескольких страниц, не ко времени изменив `CSSPath`; или, что еще хуже, в многопоточной системе, их отображение может быть испорчено другим потоком. Однако альтернативный способ, заключающийся в индивидуальном конфигурировании каждого экземпляра, будет порождать еще больше ошибок. Существуют и бо-

лее сложные решения, но они отнимают много времени на разработку конструкции и реализацию, причем с большой вероятностью можно предсказать, что такое решение приведет к еще более тяжелым и непредсказуемым результатам.

В следующей части определялись методы, содержащие базовые свойства страницы, для которых хотелось бы в будущем иметь возможность переопределения в подклассе. Лучше, чтобы языком страницы по умолчанию был английский, однако очень утомительно указывать это каждый раз. Но мы можем без затруднений сменить язык на какой-нибудь другой просто путем помещения языка в подкласс класса. (Наверное, мне следовало сначала сказать, что все это используется для заполнения тэгов <META> вверху страницы.)

Другая часть этого кода представляет собой отвлеченный метод, определяющий другое свойство, которое передается в шапку страницы, т.е. заголовок. Вот так выглядит вторая часть:

```
...
    public abstract String computeTitle();

    public String getLanguage() {
    return "en".intern();
    };

    public String getRobotInfo() {
    return "ALL";
    };

    public String getAuthor() {
    return "Olivia S. Direnzo and Paul A. Houle";

    public Date getDate() {
    return new Date();
    };

    public String getCopyright() {
    return "&copy; 1998 HoneyLocust Media Systems
        (http://www.honeylocust.com/)";
    };

    public String getKeywords() {
    return "El Limon,Weeds,Botany,xml";
    };

    public String getDescription() {
    return "A collection of descriptions and illustrations of weeds observed in El
Limon, a small village in the Dominican Republic during January 1998.";
    };

    public String getVersion() {
    return "0.95a";
    };
...
```

Другой набор методов создает тэг <HEAD> документа. Конструктивная схема создания HTML-кода ясна. Последовательность методов, называемых generateX(), фор-

мирует различные части документа. Любые повторяющиеся действия помещаются в подметод, чтобы уменьшить объем работы. В этой же части программы находится код, создающий окончание документа:

```
...
   public void generateHead() {
   d_out.println("HTML LANG=\""+getLanguage()+"\">");
   d_out.println("<HEAD><TITLE>"+computeTitle()+"</TITLE>");
   generateMetaInformation();
   generateCSSInsert();
   d_outprintln("</HEAD>");
   d_outprintln("<BODY BGCOLOR=#FFFFFF>");
   };

   public void generateMeta(String name,String content) {
   if (content==null || content.length()==0)
       return;

   d_out.println("<META NAME=\""+name+"\"CONTENT=\""+content+ "\"lang=\"
"+getLanguage()+"\">");
   };

   public void generateCSSInsert() {
   if (s_cssPath!=null) generateLink("STYLESHEET",s_cssPath,"text/css");
   };

   public void generateLink(String rel,String href) {
   if (href==null || href.length()==0)
       return;

   d_out.println("<LINK REL=\""+rel+"\" HREF=\""+href+"\">");
   };

   public void generateLink(String rel,String href,String type) {
   if (href==null || href.length()== 0)
       return;

   d_out.println("<LINK REL=\""+rel+"\" HREF=\""+href+"\
               "TYPE=\""+type+"\">");
   };

   public void generateMetaInformation() {
   generateMeta("ROBOTS",getRobotInfo());
   generateMeta("AUTHOR",getAuthor());
   generateMeta("Date",getDate().toGMTString());
   generateMeta("Copyright",getCopyright());
   generateMeta("Keywords",getKeywords());
   generateMeta("Description",getDescription());
   generateLink("Index","../common/index.html");
   generateLink("Index","../family/index.html");
   generateLink("Index","../latin/index.html");
   generateLink("Begin","../1/index.html");
   generateLink("Top","..");
   generateLink("Contents","..");
   generateLink("Start","..");
```

```
    };

    void generateTail() {
    d_out.println("<BR CLEAR=all><HR>");
    d_out.print("<I>Version "+getVersion()+" ");
    d_out.println(&copy; 1998<A HREF=\"http://www.honeylocust.com/\">Honeylocust
Media Systems.</A>");
<A HREF=\"mailto:paul@honeylocust.com\">paul@honeylocust.com</A>");
    d_out.println("</BODY></HTML>");
    };
...
```

Еще одна часть кода занимается созданием строки наверху страницы и навигационной строки внизу. Вновь я стараюсь писать небольшие методы. Один из методов, `generateNavInsert()`, умышленно сконструирован так, чтобы быть переопределяемым. Он может быть заменен методом, который добавляет рядом с кнопкой **TOP** кнопки **PREV** и **NEXT**.

```
...
    void generateNavLink(String text,String url) {
    d_out.print("<A CLASS=navlink HREF=\""+url+"\">"+text+"</A>");
    };

    void generateNavSpacer() {
    d_out.print(" | ");
    };

    public void generateNavInsert() {
    generateNavLink("TOP","../index.html");
    generateNavSpaser();
    };

    public void generateNavBar() {
    d_out.println("<BR CLEAR=ALL>");

    d_out.println(<TABLE CELLSPACING=0 WIDTH=100%><TR><TD CLASS=navbar>");
    generateNavInsert();

    generateNavLink("FAMILY","../family/index.html");
    generateNavSpacer();
    generateNavLink("LATIN","../latin/index.html");
    generateNavSpacer();
    generateNavLink("COMMON","../common/index.html");
    d_out.println("</TD></TR></TABLE>");

    };
...
```

Остальная часть программы представляет собой набор обслуживающих функций, которые созданы для того, чтобы их использовали дочерние классы класса `GenerateHTML`. Функция `outputTwoCol()` принимает вектор строк в HTML-формате и после преобразования выдает их в виде двух колонок. Функция очень проста и предполагает, что размер по вертикали пропорционален числу строк (это верно только в каком-то приближении). Но и при своей простоте функция соответствует

требованиям, предъявляемым к выполняемой ею работе, и может быть заменена на что-нибудь более совершенное (замена наследуется всеми потомками), когда возникнет такая необходимость. Функции `ensureDirectory()` и `copyFiles()` – это некоторые обслуживающие функции для работы с файлами, которые потребуются классам-потомкам; данные функции являются кандидатами на помещение в отдельный обслуживающий класс. Функция `copyFiles()` опять-таки является компромиссом между безупречными функциональными возможностями и необходимостью писать код просто и быстро. Здесь следует отметить, что поскольку функция загружает файл в память за один прием, она не может масштабироваться для копирования, скажем, 40-мегабайтного файла на моем переносном компьютере. С другой стороны, если необходимо скопировать 100-килобайтный `gif`-файл, то для такой работы она подходит в самый раз. Если способность к масштабированию станет проблемой, эту функцию тоже можно будет заменить.

```
...
    public void outputTwoCol(Vector v) {
    int half=v.size()/2;
    d_out.println("<TABLE WIDTH=100% COLS=2>");
    d_out.println("<TR><TD WIDTH=50% VALIGN=top>");
    for(int i=0;i<half;i++)
        d_out.println(v.elementAt(i).toString());

    d_out.println("<TD><TD WIDTH=50% VALIGN=top>");
    for(int i=half;i<v.size();i++)
        d_out.println(v.elementAt(i).toString());

    d_out.println("</TD></TR></TABLE>");

    };

    public String toHtml(LanguageString 1) {
    if (l.getLanguage()==getLanguage()) {
        return l.toString();
    } else {
        return "<SPAN LANG=\""+l.getLanguage()+"\">"+l.toString()+"</SPAN>";
    };
    };

    public static File ensureDirectory(File dir,String s) throws IOException{
    return ensureDirectory(new File(dir,s));
    };

    public static File ensureDirectory(File f) throws IOException{
    if (f.exists()) {
        if (!f.isDirectory()) throws new IOException("File"+f.toString()+
            "already exists and is not a directory.");

        return f;
    };

    f.mkdirs();
    return f;
```

```
    };

    public static void copyFile(File from,File to) throws IOException {
    FileInputStream inStream=new FileInputStream(from);
    FileOutputStream outStream=new FileOutputStream(from);
    byte buff[]=new byte[inStream.available()];
    inStream.read(buff);
    outStream.write(buff);
    inStream.close();
    outStream.close();
    };

};
```

Класс GeneratePlant.java

Теперь, когда мы представляем себе, какая инфраструктура должна быть построена, можно приступить к формированию кода, создающего описания растений. Это `GeneratePlant.java`:

```
package honeylocust.limon;

import honeylocust.limon.representation.AnchorChunk;
import honeylocust.limon.representation.LanguageString;
import honeylocust.limon.representation.Species;
import honeylocust.limon.representation.RefChunk;
import honeylocust.limon.representation.Text;
import honeylocust.limon.representation.TextChunk;

import java.io.PrintWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.FileNotFoundException;

public class GeneratePlant extends GenerateHtml {
    PlantIndex d_index;
    species d_species;
    File d_dir;

    public GeneratePlant(PlantIndex index,Species s) throws IOException {
    super(toHtmlPath(s.getId()),"index.html");
    d_dir=ensureDirectory(toHtmlPath(s.getId()));
    d_species=s;
    d_index=index;
    };
...
```

Поскольку необходимо, чтобы класс `GeneratePlant` ссылался на индекс, он получает его в конструкторе.

Сейчас самое время приступить к переопределению некоторых методов, чтобы обеспечить правильную метаинформацию для описаний растений. Переопределяется даже метод `generateMetaInformation` – чтобы обеспечить два дополнительных тэга `<META>` (`PREV` и `NEXT`), которые не во всех документах будут уместны.

```
...
    public String computeTitle() {
    return computeTitle(d_species);
    };
    private String computeTitle(Species s) {
    if (!s.identified()) {
        return (s.getId()+"Not Identified");
    };
    String latin=s.getLatin()[0].toString();
    Integer myNumber=d_index.plantNumber(s);
    String number=(myNumber!=null) ? myNumber.toString()+" " : " ";
    return number+latin;
    };
    public String getKeywords() {
    StringBuffer sb=new StringBuffer(super.getKeywords());
    LanguageString family=d_species.getFamily();
    if (family!=null) {
        sb.append(',');
        sb.append(family.toString());
    };
    if (d_species.identified()) {
        LanguageString latin[]=d_species.getLatin();
        for(int i=0;i<latin.length;i++) {
            sb.append(',');
            sb.append(latin[i].toString());
        };
        LanguageString common[]=d_species.getCommon();
        for(int i=0;i<common.length;i++) {
            sb.append(',');
            sb.append(common[i].toString());
        };
    };
    return sb.toString();
    };
    public void generateMetaInformation() {
    super.generateMetaInformation();
    Integer myNumber=d_index.plantNumber(d_species);
    boolean numeric=((myNumber!=null) && d_index.isNumericId());
    if (numeric && myNumber.intValue() > d_index.getMinPlantNumber()) {
        generateLink("Prev","../"+(myNumber.intValue()-1)+"/index.html");
    };
    if (numeric && myNumber.intValue() < d_index.getMaxPlantNumber()) {
        generateLink("Next","../"+(myNumber.intValue()+1)"/index.html");
    };
    };
...
```

В этом месте мы переопределяем `generateNavInsert()`, так чтобы он включал ссылки на предыдущую и последующую страницы, если таковые существуют, и вызывал код, создающий описания растений:

```
...
/* Включение ссылок на предыдущую и последующую страницы. */
    public void generateNavInsert() {

    Integer myNumber=d_index.plantNumber(d_species);

    boolean numeric=((myNumber!=null) && d_index.isNumericId());

    if (numeric && myNumber.intValue() > d_index.getMinPlantNumber()) {
        generateNavLink("PREVIOUS","../"+(myNumber.intValue()+1)+"/index.html");
        generateNavSpacer();
    };
    super.generateNavInsert();
    if (numeric && myNumber.intValue() < d_index.getMinPlantNumber()) {
        generateNavLink("NEXT","../"+(myNumber.intValue()+1)+"/index.html");
        generateNavSpacer();
    };
    };

/* Вызов кода, создающего описания растений. */
    public synchronized void generate(Species s) throws
IOException,InterruptedException {

    File dir=ensureDirectory(toHtmlPath(s.getId()));

    PrintWriter w=new PrintWriter(new FileWriter(new File(dir,"index.html")));
    setSpecies(s);

    copyImages(dir,s);

    try {
        generateHead(w);
        generateTopBar(w);
        generateInLine(w,s,dir);
        generateFamily(w,s);
        generateLatin(w,s);
        generateCommon(w,s);
        generateDescriptions(w,s);
        generateAdditional(w);

        generateNavBar(w);
        generateTail(w);
    }
    catch(GenerateHtmlException e) {
        System.err.println(">> Error Generating HTML for Species "+s.getId());
        System.err.println(">> "+e.toString());
        System.err.println();

    };

    w.close();

};
```

```
    public void generateInLine(PrintWriter w,Species s,File dir) throws Interrupte
dException,FileNotFoundException {
    WebImage wi=new WebImage(dir,"small.gif");
    w.println("<A HREF=\"big.gif\">");
    wi.setAlign("RIGHT");
    wi.setAlt('['+d_index.speciesName(s).toString()+" illustration]");
    wi.setBorder(false);
    wi.insert(w);
    w.println("</A>");
    };

    public void generateFamily(PrintWriter w,Species s) {
    if (s.getFamily()==null) {
        w.println("<SPAN CLASS=unknown>Family Unknown</SPAN>");
    }
    else {
        w.println("<B>Family "+s.getFamily()+"</B>");
    };
    };

    public void generateLatin(PrintWriter w,Species s) {
    LanguageString latin[]=s.getLatin();
    if (latin.length==0) {
        w.println("<BR><SPAN CLASS=unknown>Species not identified</SPAN>');
    } else {
        w.println("<BR>Latin name: <I>"+toHtml(latin[0])+"</I>");
    }
    };

    public void generateCommon(PrintWriter w,Species s) {
    LanguageString common[]=s.getCommon();
    if (common.length==0)
        return;

    if (common.length==1) {
        w.println("<BR>common name: <I>");
    } else
        w.println("<BR>Common names: <I>");

    for(int i=0;i<common.length;i++) {
        w.print(toHtml(common[i]));
        if (i<common.length-1)
            w.print(", ");
    }

    w.println("</I>");
    };

    public void generateDescriptions(PrintWriter w,Species s) throws
GenerateHtmlException {

    Text texts[]=s.getTexts();
    for(int i=0;i<texts.length;i++)
        if (selectText(texts[i]))
```

```
            generateText(w,texts[i]);
    };

    boolean selectText(Text t) {
    return true;
    };

    void generateText(PrintWriter w,Text t) throws GenerateHtmlException {
    w.println("<P>");
    TextChunk[] chunks=t.getChunks();
    for(int i=0;i<chunks.length;i++) {
        generateChunk(w,chunks[i]);
    };
    };

    void generateChunk(PrintWriter w,TextChunk c) throws GenerateHtmlException {
    if (c instanceof AnchorChunk) {
        generateAnchorChunk(w,(AnchorChunk) c);
    } else if (c instanceof RefChunk) {
        generateRefChunk(w,(RefChunk) c);
    } else
        w.println(c.getText());
    };

    void generateAnchorChunk(PrintWriter w,AnchorChunk ac) {
    w.println("<A HREF=\""+ac.getHref()+"\">"+ac.getText()+"</A>");
    };

    void generateRefChunk(PrintWriter w,RefChunk ac) throws GenerateHtmlException
{
    Species target=d_index.getById(ac.getI());
    if (target==null)
        throw new GenerateHtmlException("Invalid ref id.");

    w.println("<A HREF=\"../"+ac.getId()+"/index.html\">"+d_index.
speciesName(target)+"</A>");
    };

    public void generateAdditional(PrintWriter w) {
    Species s=d_species;
    w.println("<P><B>Additional resources:</B>\n");
    w.println("<UL>\n");
    w.println("<LI><A HREF=\"big.gif\">Large botanical illustration</A>\n");
    w.println("</UL>\n");

    };

    public void copyImages(File dir,Species s) throws IOException {
    File bigImg=new File(dir,"big.gif");
    File smallImg=new File(dir,"small.gif");
    copyFile(s.getBigImage(),bigImg);
    copyFile(s.getSmallImage(),smallImg);
    };
...
```

Здесь я решил использовать новые возможности JDK 1.1, которые облегчают

составление собственных исключений. Класс `GenerateHtmlException` является внутренним классом, это новая возможность JDK 1.1. Внутренним классом называется класс, определяемый внутри другого класса и являющийся видимым только внутри данного класса. В языке Java поощряется частое создание программистами новых классов, например, каждый раз, когда требуется определить новый тип исключения или когда необходимо написать обработчик событий в событийной модели JDK 1.1. Если определяется совершенно новый класс и создается новый `java`-файл для каждого нового класса, в итоге может возникнуть большой беспорядок, устранимый разве что путем использования внутренних классов. Официальная спецификация внутренних классов находится на странице

http://java.sun.com/products/jdk/1.1/docs/guide/innerclasses/spec/innerclasses.doc.html/

```
...
   class GenerateHtmlException extends Exception {
   public GenerateHtmlException(String s) {
       super(s);
   };
   };
};
```

Класс GenerateIndex.java

Как `GenerateHtml` предоставляет основу для построения на ней обобщенных Web-страниц, так и `GenerateIndex` предоставляет основу для построения на ней страниц указателя. Переопределяя всего несколько ключевых методов, я могу создавать два типа указателей, один для названий семейств и другой для народных названий

```
package honeylocust.limon;

import honeylocust.limon.representation.LanguageString;
import honeylocust.limon.representation.Species;

import java.io.IOException;
import java.io.File;
import java.util.Vector;
import netscape.util.Sort;

public abstract class GenerateIndex extends GenerateHtml {

   PlantIndex d_index;
   boolean p_addLetters;

   public GenerateIndex(PlantIndex index,File dir) throws IOException {
   super(dir,"index.html");
   d_index=index;
   p_addLetters=false;
   };
...
```

Следующие два метода используют конструкционную схему для аксессора в структуре Java Beans. Это позволяет другим объектам легче конфигурировать поведение данного объекта. Конкретно, опция `setAddLetters` предоставляет воз-

можность создания высокой заглавной буквы в начале секции указателя, начинающейся с новой буквы алфавита.

```
...
   public void setAddLetters(boolean addLetters) {
   p_addLetters=addLetters;
   };

   public boolean getAddLetters() {
   return p_addLetters;
   };
...
```

Два следующих метода переопределяются, для того чтобы принять решение, какой тип указателя вырабатывать. Метод `getKeys()` – это список позиций в указателе, а `getSpecies()` получает виды растений, которые согласуются с конкретным ключом.

```
abstract LanguageString[] getKeys();

abstract Species getSpecies(LanguageString key);

public synchronized void generate() throws IOException,InterruptedException{

generateHead();
generateTopBar();
generateIndex();
generateNavBar();
generateTail();

d_out.close();
};

public void generateIndex() {
LanguageString keys[]=getKeys();
Sort.sort(keys,null,0,keys.length,true);
Vector entries=new Vector();
char letter='';
for(int i=0;i<keys.length;i++) {
   Species s=getSpecies(keys[i]);
   char nextLetter=Character.toUpperCase(keys[i].toString().charAt(0));
   String prefixString="";

   if (p_addLetters && nextLetter>letter)
       prefixString="<BIG>"+nextLetter+"</BIG><BR>";

   letter=nextLetter;

   entries.addElement(prefixString+"<A HREF=\"../"+s.getId()+"/index.html\"
       >"+toHtml(keys[i])+"</A><BR>")
   };
   outputTwoCol(entries);
   };
};
```

Класс GenerateLatin/Common.java

На этом этапе создание конкретного указателя выполняется простым заполнением пропусков.

```
package honeylocust.limon;

import honeylocust.limon.representation.LanguageString;
import honeylocust.limon.representation.Species;
import java.io.IOException;

public class GenerateLatin extends GenerateIndex {

    public generateLatin(PlantIndex index) throws IOException {
    super(index,toHtmlPath("latin"));
    };

    public String computeTitle() {
    return "Index by Latin Name";
    };

    LanguageString[] getKeys() {
    return d_index.getLatinNames();
    };

    Species getSpecies(LanguageString k) {
    return d_index.getByLatin(k);
    };

};

package honeylocust.limon;

import honeylocust.limon.representation.LanguageString;
import honeylocust.limon.representation.Species;
import java.io.IOException;

public class GenerateCommon extends GenerateIndex {

    public generateCommon(PlantIndex index) throws IOException {
    super(index,toHtmlPath("common"));
    };

    public String computeTitle() {
    return "Index by Common Name";
    };

    LanguageString[] getKeys() {
    return d_index.getCommonNames();
    };

    Species getSpecies(LanguageString k) {
    return d_index.getByCommon(k);
    };

};
```

Класс GenerateFamily.java

Указатель по семействам радикально отличается от двух других, поскольку представляет собой двухуровневую иерархию. В семейство может входить более чем один вид. Мы намерены отобразить вместе все растения, принадлежащие одному семейству, и хотим, чтобы они были упорядочены сначала по семействам, а затем по видам. Это осуществляется с помощью приведенного далее кода.

```
package honeylocust.limon;

import honeylocust.limon.representation.LanguageString;
import honeylocust.limon.representation.Species;

import com.sun.java.util.collections.Arrays;
import java.io.IOException;
import java.util.Vector;
// import netscape.util.Sort.

public class GenerateFamily extends GenerateHtml {

   PlantIndex d_index;

   public GenerateFamily(PlantIndex index) throws IOException {
   super(toHtmlPath("family"),"index.html");
   d_index=index;
   };

   public String computeTitle() {
   return "Index by Family";
   };

   public synchronized void generate() throws IOException,
               InterruptedException {

   generateHead();
   generateTopBar();
   generateIndex();
   generateNavBar();
   generateTail();

   d_out.close();

   };

   public void generateIndex() {
   LanguageString keys[]=d_index.getFamilies();
   Sort.sortStrings(keys,0,keys.length,true,true);
   Vector entries=new Vector();

   for(int i=0;i<keys.length;i++) {
       StringBuffer sb=new StringBuffer();
       sb.append("<B>"+toHtml(keys[i])+"</B>\n");
       sb.append("<UL TYPE=\"square\">\n");
       Species s[]=d_index.getByFamily(keys[i]);
       for(int j=0;j<s.length;j++) {
       sb.append("<LI><A HREF=\"../"+s[j].getId()+"/index.html\">" +toHtml(d_
index.speciesName(s[j]))+"</A>n");
       };
```

```
            sb.append("</UL>");
            entries.addElement(sb);
        };
        outputTwoCol(entries);
        };
};
```

Как самому построить приложение «Сорняки Эль Лимона»

Приведенные ниже наборы инструкций описывают установку только на платформах Windows и Unix.

Построение «Сорняков...» под Windows

Для начала необходимо загрузить соответствующие `zip`-файлы с кодом для проекта «Сорняки...» с одного из следующих Web-сайтов:

http:/honeylocust.com/limon/xml/index.html
http://webdev.wrox.co.uk/books/1525

Далее потребуется копия Sun Java 1.1.7 JDK для Windows, которая может быть бесплатно получена с Web-сайта

http://java.sun.com/products/jdk/

Установите JDK в соответствии с инструкцией, включив путь

```
C:\JDK1.1.7\BIN
```

в инструкцию `PATH` в файле `autoexec.bat` в корневой директории жесткого диска, например, так:

```
PATH C:\;C:\WINDOWS;C:\WINDOWS\COMMAND;C:\DOS;C:\JDK1.1.7\BIN;
```

и если папка `classes` в структуре директорий еще не существует, создайте папку `classes` в основной папке JDK 1.1.7.

Разархивируйте Java-классы «Сорняков...» из файлов `ifcUtil.jar`, `hyMSXML.jar` и `limon.jar` в директорию `classes`, которую вы создали в папке JDK 1.1.7. Затем распакуйте файлы `plantxml.zip` и `images.zip` в ту же директорию `classes`. Когда вы закончите эти операции, у вас должна получиться структура директорий, представленная на рис. 12.4.

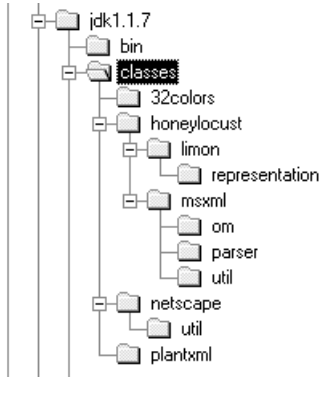


Рис. 12.4. Структура директорий при самостоятельном построении приложения «Сорняки...» под Windows

Вы можете либо отредактировать файл `autoexec.bat`, чтобы установить переменную окружения CLASSPATH, вставив следующую строку

```
Set CLASSPATH=%CLASSPATH%;C:\JDK1.1.7\CLASSES;
```

либо набрать ту же самую команду в командной строке DOS в окне MS-DOS. Если вы отредактировали файл `autoexec.bat`, вам может понадобится перезагрузить свой компьютер, чтобы изменения вступили в силу. Если вы набрали эту строку в сеансе DOS, то переменная CLASSPATH останется установленной только до тех пор, пока открыто окно DOS.

В окне DOS перейдите в директорию `C:\JDK1.1.7\CLASSES` и наберите в командной строке DOS

```
java honeylocust.limon.Generator plantxml\*.xml
```

При этом приложение должно создать основной набор Web-страниц и поместить их в папку `output`, находящуюся в папке `C:\JDK1.1.7\classes`.

Для полного эффекта необходимо разархивировать содержимое файла `output.zip` в папку `output`. Если затем вы запустите Проводник Windows и дважды щелкнете на файле `index.html` в папке `output`, ваш браузер должен запуститься и показать основной указатель проекта «Сорняки...». В качестве альтернативы можно запустить браузер и открыть файл `index.html`, воспользовавшись пунктом меню (**File ⇒ Open**).

Построение «Сорняков...» под UNIX

Для начала необходимо загрузить соответствующие `.zip` файлы с кодом для проекта «Сорняки...» с одного из следующих Web-сайтов:

http:/honeylocust.com/limon/xml/index.html
http://webdev.wrox.co.uk/books/1525

Затем вам потребуется самая свежая версия Java 1.1.x Development Kit (JDK), имеющаяся в наличии для вашей платформы. Она доступна на Web-сайтах поставщиков системы UNIX. Адреса поставщиков некоторых популярных версий UNIX приведены в табл. 12.1.

Таблица 12.1. Адреса поставщиков Java JDK для некоторых популярных версий UNIX

Операционная система	Web-сайт для JDK
Solaris	http://java.sun.com/products/jdk/
Linux	http://www.blackdown.org/java-linux/html
Digital	http://www.digital.com/java/
AIX	http://www.ibm.com/java/tools/jdk.html

Установите JDK в соответствии с инструкцией. Затем распакуйте файл `limon.zip` в вашу директорию `home`. Вы можете сделать это, набрав команды:

```
% cd $HOME
% unzip limon.zip
```

Затем необходимо установить CLASSPATH, чтобы Java мог найти файлы `class`, которые составляют пакет `'limon'`. Чтобы сделать это из `csh` или другой совместимой с ней оболочки, наберите:

```
% setenv LIMON ~/limon
% setenv CLASSPATH
$LIMON/ifcUtil.jar:$LIMON/hnyMSXML.jar:$LIMON/limon.jar:$ CLASSPATH
```

Если вы используете оболочку, происходящую от Bourne shell, например, `sh` или `bash`, наберите:

```
% export LIMON=$HOME/limon
% export
CLASSPATH=$LIMON/ifcUtil.jar:$LIMON/hnyMSXML.jar:$LIMON/limon.jar:$CLASSPATH
```

Если вы предпочтете распаковать файл `limon.zip` в другую директорию, не забудьте обновить переменную окружения `$LIMON`, чтобы отразить этот факт. Вы можете пожелать поместить вашеуказанные строчки в ваш `~/.login`, чтобы они автоматически выполнялись, когда вы входите в систему.

Затем наберите:

```
% cd $LIMON
% unzip plantxml.zip
% unzip images.zip
% unzip output.zip
```

чтобы распаковать поддерживающие файлы. И наконец, наберите:

```
% java honeylocust.limon.Generator plantxml\*.xml
```

чтобы создать Web-страницы и поместить их в директорию `$LIMON/output`.

Заключение

Надеемся, нам удалось наглядно продемонстрировать, что язык XML вполне пригоден для разделения стиля и содержания. Этот метод дает возможность строить более крупные проекты, чем те, которые можно выполнить вручную. Авторы применили XML для подготовки своеобразного гербария, в котором описания растений выполнены на XML, а Java-приложение использовали для преобразования описаний в HTML. Как было показано, этот механизм, давая возможность управлять каждой частью программы по отдельности, помог улучшить внешний вид страниц и их совместимость с устаревшими и малораспространенными Web-браузерами. Действующий продукт, который был создан в сравнительно короткие сроки, легко приспособить к будущим требованиям и модификациям программного обеспечения.

Глава 13. Формат определения канала

Одна из областей Web-разработок, на которую расширяемый язык разметки (XML) оказал огромное влияние, — это *активные каналы* (Active Channels) для браузера Internet Explorer 4.x (CDF-технология не поддерживается браузером Netscape Communicator 4). Именно язык XML стоит за командным центром каналов — файлом *формата определения каналов* (Channel Definition Format, CDF). По существу, CDF-технология является приложением языка XML.

Активный канал, в самом общем виде, – это набор Web-страниц и ассоциированных с ними файлов (таких как изображения и звуковые выводы), а также CDF-файл. Web-страницы образуют информационную часть канала, в то время как CDF-файл, условно говоря, предоставляет в распоряжение пользователей особый компоновщик канала, обладающий гибкими и простыми в использовании методами, обеспечивающими доступ к информационной части. Всего лишь одним нажатием клавиши посетители (или, как их часто называют, подписчики) могут загрузить всю информацию по сети и локально поместить ее в кэш браузера IE4, и при этом получить навигационную иерархию, созданную в *подокне каналов* (Channel Pane) браузера. После этого подписчик, работая в автономном режиме (offline), может просматривать всю информацию с большей скоростью и меньшими затратами, чем при работе в режиме активного соединения с Internet.

CDF-файл, созданный для содержимого канала, является по сути дела метаданными, то есть данными о данных. Как таковой, CDF-файл не несет в себе содержимого — он является «картой сайта», которая создается для того, чтобы подписчики могли быстрее и легче ориентироваться в его топологии и просматривать информацию. CDF-файл использует XML для управления основными областями развертывания активного канала и последующим обновлением информации. Короче говоря, CDF-файл выполняет следующие функции:

- управление тем, что составляет содержание канала; это могут быть обычные HTML-страницы, заставки (screensaver), активные компоненты рабочего стола (Active Desktop Components);
- определение, каким образом браузеру IE4 следует использовать страницы (возможны четыре варианта: нормальный вид, полноэкранный вид, в качестве заставки или в качестве компонента рабочего стола);
- выбор режима; должно ли быть загружено все содержимое канала для просмотра в автономном режиме или достаточно создать навигационную иерархию для облегчения ориентации при работе в диалоговом (online) режиме;

- управляение формированием навигационной иерархии в подокне каналов браузера IE4;
- осуществление индивидуализации канала при помощи логотипов, иконок, заголовков и аннотаций;
- сохранение и своевременное использование расписания, согласно которому должно осуществляться регулярное обновление содержания канала.

Невольно возникает вопрос, неужели вся эта информация помещается в одном маленьком CDF-файле? (Обычно его размер находится в диапазоне 1-2 Кбайт для канала, состоящего из 10-15 документов, хотя, как вы увидите далее, чем больше позиций вы помещаете в CDF-файл, тем более объемистым он становится.) Благодаря универсальности языка XML, который используется для организации подобных механизмов управления, ответ на этот вопрос будет положительным. CDF-файлы невелики и их легко создавать. Для этой цели вам не потребуется никакого другого сложного оборудования, кроме вашего любимого текстового редактора (простой Блокнот в среде Windows идеально подходит для написания CDF-файлов). Перед тем как приступить к созданию CDF-файла, давайте рассмотрим ситуацию, в которой для повышения эффективности существующего Web-сайта использована CDF-технология; при этом необходимость изменения какой-либо из уже существующих Web-страниц просто не возникает.

Уточнение *Следует заметить, что в настоящий момент активные каналы разработаны для просмотра в автономном (offline) режиме. Технология добавления CDF-файла к уже существующему Web-сайту даст великолепные результаты, если на стороне сервера не используются такие компоненты, как CGI, ASP или Microsoft FrontPage Extension. Если вы хотите создать канал из сайта, где задействованы эти компоненты, вам либо придется сделать ваш канал работающим в диалоговом (online) режиме, либо так изменить код Web-страниц, чтобы использовать технологии на стороне клиента, например, исполнение сценариев (VBScript и JavaScript) и привязку данных (databinding).*

Учебный пример на CDF–технологию

Мы собираемся рассмотреть вымышленную компанию – корпорацию XYZ Corp, – и посмотреть, что она могла бы выиграть, воспользовавшись преимуществами CDF-технологии для создания активного канала.

Исходная ситуация

Корпорация XYZ Corp специализируется на изготовлении всевозможной мелочи (widgets), типа разнообразных кнопок, пуговиц, брелков, клепок на одежду и т.п., для постоянного растущего рынка таких изделий (добрые старые штучки!). У корпорации XYZ Corp уже есть свой собственный Web-сайт, который используется для двух целей:

- каталог, применяемый при работе в диалоговом (online) режиме, обеспечивает клиентам мгновенный доступ ко всему спектру выпускаемых изделий и цен на них. При этом снижаются затраты на распечатывание и распространение бумажных каталогов, информирование потребителей обо всех изделиях происходит быстрее;
- каталог со специальными предложениями, работа с которым ведется в диалоговом (online) режиме. Используется коммивояжерами корпорации XYZ Corp. При помощи этого каталога они имеют мгновенный доступ к информации по полному спектру изделий и цен на них. Коммивояжеры пользуются переносными компьютерами и сотовыми телефонами для подключения к Web.

Корпорация XYZ Corp обновляет свой основной каталог и страницы со специальными предложениями каждую пятницу в середине дня. У корпорации также есть страница **Whats New** (Что нового), перечисляющая новые изделия и специальные предложения, и страница **Whats Happening at XYZ** (Что происходит в корпорации XYZ), которая обеспечивает как клиентов, так и сотрудников информацией о деятельности корпорации и ее новостях. На сайте дополнительно расположено несколько других страниц, на которых публикуются сведения о сроках и условиях сделок для клиентов и информация о гарантийных обязательствах и сервисном обслуживании.

В настоящее время, чтобы получить обновленный каталог, выпускаемый еженедельно, клиенты постоянно должны помнить (либо создавая закладки, либо полагаясь на память или записи, либо получая по электронной почте уведомление об изменении страниц, в котором содержится соответствующий URL-адрес) о необходимости посещения нужной страницы. Клиенты вынуждены обращаться к каталогу каждый раз, когда хотят с ним свериться; при этом тратится рабочее время, снижается эффективность работы, растут расходы на оплату времени соединения с сетью.

Корпорация XYZ Corp хотела бы сделать свои услуги, предоставляемые в диалоговом режиме, более привлекательными для клиентов. С этой целью руководство решило:

- обеспечить пользователям доступ к Web-сайту без необходимости запоминания URL-адреса или создания закладок, увеличивая, таким образом, использование сервиса, предоставляемого в диалоговом режиме (online сервиса);
- повысить эффективность показателя «время на экран» (time to screen), то есть скорость, с которой пользователи могут просматривать и использовать каталог, предназначенный для работы в диалоговом режиме. Продуктивность безусловно повысится, поскольку клиенты будут получать данные из кэша на своем жестком диске, а не непосредственно из Internet;
- гарантировать, что у клиентов под рукой всегда находится самая свежая информация о продукции, ценах и услугах.

Каким же образом можно использовать CDF-технологию для решения этих задач?

Какую пользу принесет использование CDF–технологии

Применение CDF-технологии для создания активного канала дает возможность тотчас взяться за работу по всем трем направлениям, не затрачивая времени, усилий или денежных средств на какое-либо реальное изменение топологии Web-сайта. Никто не отрицает, что корпорация XYZ Corp могла бы создать специализированный набор страниц исключительно для использования в канале; набор, в котором можно воспользоваться другими полезными технологиями, предоставляемыми браузером IE4 – такими, например, как DHTML и привязка данных. Однако авторы сознательно ограничиваются рамками CDF-технологии для реализации замыслов корпорации XYZ Corp. В этом случае:

- клиентам больше не придется запоминать URL-адреса страниц, которые они хотят посетить. У них будет простой доступ к узлу. При помощи всего одного щелчка в *подокне каналов* (chanel pane) браузера IE4 или на *панели каналов* (channel bar) рабочего стола они смогут добраться до нужного адреса (рис. 13.1 и 13.2 соответственно);

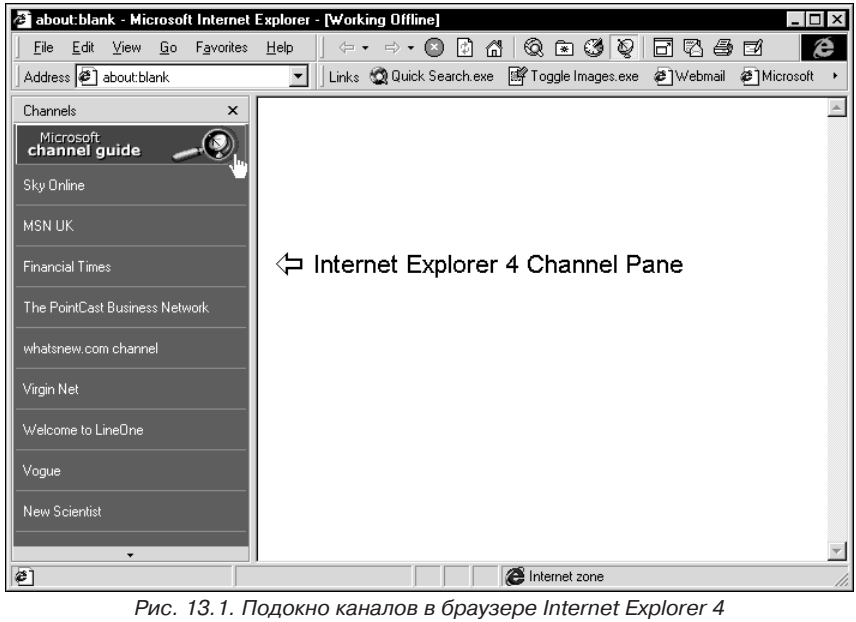


Рис. 13.1. Подокно каналов в браузере Internet Explorer 4

- CDF-файл можно настроить таким образом, что каталог и другие соответствующие документы будут загружены из Сети и помещены в локальный кэш браузера подписчика. При этом информация мгновенно становится доступной подписчику, отпадает необходимость выхода в Сеть и повторной загрузки документов. Это значительно снижает нагрузку на сервер, и к тому же помогает вытеснить бумажную версию каталога для клиентов, пользующихся браузером IE4. Уменьшение затрат на печать и распространение каталога – экономия весомая; она вполне может отразиться на снижении цены продукции, делая ее более привлекательной для клиентов;

- если расписание внесения изменений в канал выставлено так, что совпадает с днем и временем обновления каталога, это означает, что браузер IE4 будет автоматически загружать из Сети содержание канала и в распоряжении клиентов (на их компьютерах) будет самая последняя версия каталога. Такая настройка системы гарантирует, что клиенты будут обладать самой свежей информацией об ассортименте продукции и ценах на нее.

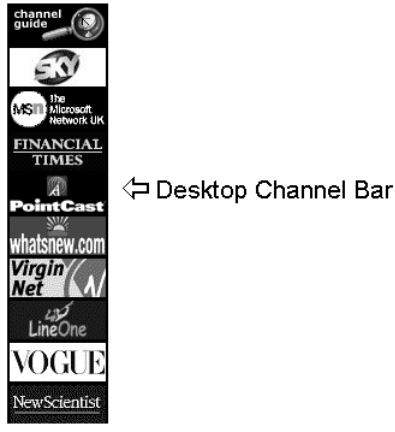


Рис. 13.2. Панель каналов на рабочем столе

Создание активного канала также принесет огромную пользу коммивояжерам, которым необходим постоянный доступ к информации в каталоге. CDF-файл позволит браузеру IE4 обновлять информацию канала раз в неделю и хранить данные локально на переносных компьютерах в кэше временных файлов Internet, устраняя, таким образом, необходимость подключаться к Internet для доступа к каталогу в промежутках между обновлениями содержания. Это существенно снизит затраты коммивояжеров на оплату времени соединения по сотовым телефонам и впечатляюще увеличит скорость вывода информации на экран (обычная скорость соединения по сотовой связи составляет 9600 килобит в секунду в стандарте GSM 900 и 1800, хотя в системах GSM 1900 доступны и более высокие скорости).

Способ, который фактически использует браузер IE4 для обновления содержания, зависит от типа соединения пользователя с Internet. Пользователи с постоянным подключением к Internet (например, по выделенной линии) обнаружат, что содержание обновляется через регулярные интервалы в те моменты, когда они не пользуются своими компьютерами (цель, преследуемая подобной настройкой браузера IE4, — снижение загрузки системных ресурсов). У пользователей с доступом к Internet по коммутируемой линии (dial-up access) есть два варианта обновления содержания:

- в первом варианте браузер IE4 автоматически дозванивается, устанавливает соединение с Internet и проверяет наличие изменений в содержимом канала (это распространяется только на пользователей, подключающихся к Internet при помощи коммутационной сети Microsoft Windows (Microsoft Windows Dial-up Networking));
- во втором варианте браузер IE4 проверяет наличие изменений в то время, когда пользователи работают с другими узлами в Internet. Это происходит незаметно: пользователи не уведомляются о том, что идет процесс обновления.

Создание CDF-файла

Итак, наступил момент, когда можно перейти к созданию CDF-файла, способного значительно улучшить существующий Web-сайт.

Перед тем как открыть Блокнот, необходимо потратить несколько минут на изучение страниц, которые корпорация XYZ Corp хочет включить в канал. В табл. 13.1 дан список таких страниц с их URL адресами:

Таблица 13.1. Страницы, которые требуется включить в канал

Страница	URL
Main Page (Главная страница)	http://www.xyzcorp.com/index.htm
Product Catalog (Каталог продукции)	http://www.xyzcorp.com/catalog.htm
Special Offers (Специальные предложения)	http://www.xyzcorp.com/special.htm
What's New (Что нового)	http://www.xyzcorp.com/new.htm
What's Happening at XYZ (Что происходит в корпорации XYZ)	http://www.xyzcorp.com/happening.htm
Terms and Conditions (Сроки и условия)	http://www.xyzcorp.com/terms.htm
Warranty (Гарантия)	http://www.xyzcorp.com/warranty.htm

Теперь самое время приготовить Блокнот и приступить к написанию XML. Файл следует сохранить с расширением `CDF`. Чтобы справиться с привычкой Блокнота сохранять файлы с расширением `txt`, необходимо указать полное имя файла в кавычках.

Начать можно с помещения в CDF-файл некоторой основы:

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

</CHANNEL>
```

Здесь имеются два базовых элемента, которые образуют CDF-файл. Первый элемент декларирует, что данный документ является XML-документом и определяет используемую версию XML и кодировку набора символов. Следует отметить, что это единственный элемент CDF-формата, в котором используется знак вопроса (`?`) в ограничителе – во всех все остальных элементах используется косая черта (`/`). Второй элемент – это элемент `CHANNEL` (Канал), внутри которого будут помещены описания информации канала и подканалов.

К настоящему моменту использовано два типа элемента из трех, перечисленных в табл. 13.2, которые будут встречаться при создании CDF-файлов.

Таблица 13.2. Типы элементов, используемые при создании CDF-файлов

Тип элемента	Пример
Пустой элемент с атрибутами	<ELEMENT ATTR1="value" ATTR2="value" />
Элемент с содержимым и закрывающим тэгом	<ELEMENT>Содержимое элемента</ ELEMENT>
Родительский элемент с атрибутами и дочерними элементами	<PARENT ATTR1="value"> <CHILD1> </CHILD1> <CHILD2 ATTR1="value" /> <PARENT/>

Как избежать ошибок в CDF-файле

Таким образом, нам уже встретился пример пустого элемента с атрибутами (`<?xml version="1.0" encoding="UTF-8"?>`). Вскоре станет ясно, что элемент `<CHANNEL>` является родительским элементом с атрибутами и дочерними элементами. Тем, кто знаком с HTML, возможно, трудно будет привыкнуть к элементам без закрывающего тэга, но следует помнить, что это XML, а не HTML. Об этом никогда не стоит забывать и каждый раз пользоваться правильным синтаксисом для закрывающих тэгов, так как неверное завершение элемента приведет к возникновению ошибок при подписке на канал.

Ошибки в CDF-файле могут выявиться на разных стадиях: во время и после подписки на канал. Диалоговое окно (рис. 13.3) отображает сообщение об ошибке во время процесса оновления содержания.

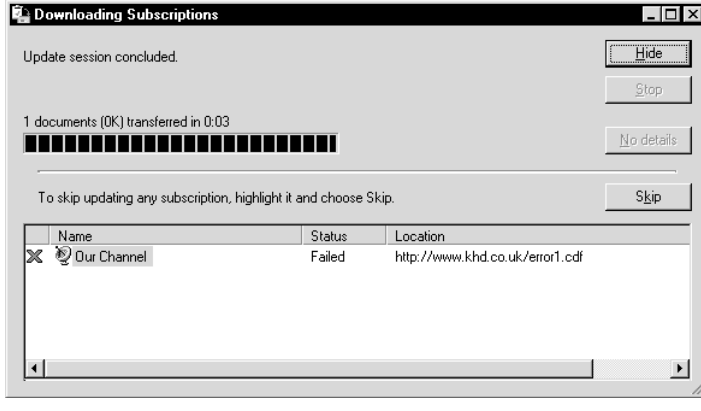


Рис. 13.3. Сообщение об ошибке во время процесса обновления содержания

Следующий экран (рис. 13.4) – это диалоговое окно **Управление подпиской** (Manage Subscription) в браузере IE4 (доступное через пункты меню **Избранное** ⇒ **Управление подпиской** (Favorites ⇒ Manage Subscription)), показывающее, что ошибка возникла во врем обновления канала:

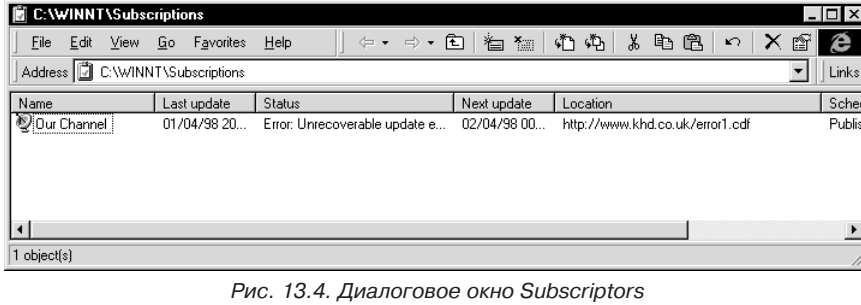


Рис. 13.4. Диалоговое окно Subscriptors

А дальше и вовсе ужасно! Когда пользователь пытается просмотреть канал с неверным CDF-файлом, он видит диалоговое окно, сообщающее, что существует проблема с написанием канала. Вот это да! По крайней мере сообщение об ошибке все же дает информацию о том, где CDF-анализатор наткнулся на шибку в CDF-файле.

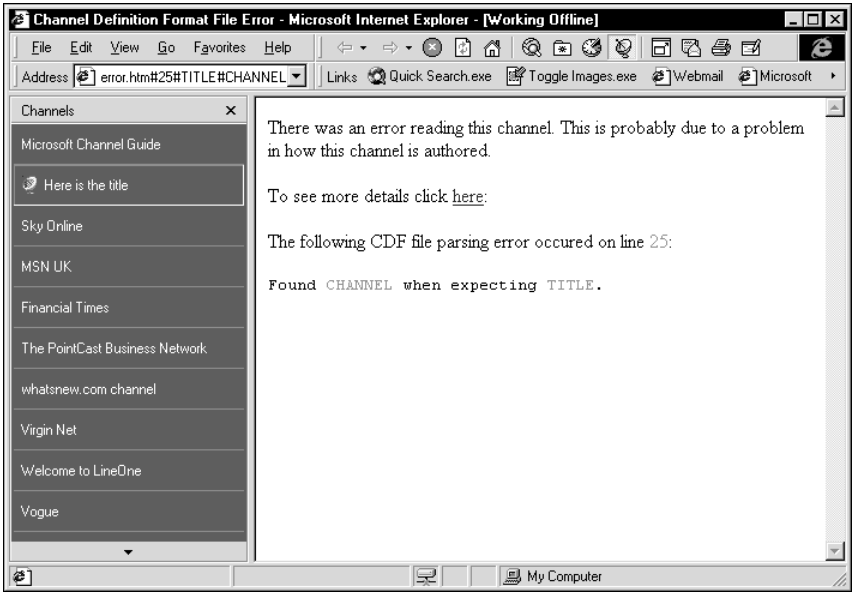


Рис. 13.5. Сообщение об ошибке при попытке чтения канала с неверным CDF-файлом

Уточнение *При попытке реализовать этот пример на практике, в зависимости от того, какая у вас конструкция браузера IE4, вам, возможно, придется удалить существующую версию канала корпорации XYZ Corp перед тем, как опробовать новую версию, в противном случае обновление новой версии может происходить неверно. Удалить существующую версию канала проще всего, щелкнув правой клавишей мыши на названии канала в подокне каналов браузера IE4 и выбрав «удалить». Затем следует перезагрузить канал со всеми его новыми характеристиками.*

Вернемся к созданию CDF-файла

Если вновь взглянуть на URL-адреса страниц, которые необходимо включить в канал, можно обнаружить, что все они находятся в корне `www.xyzcorp.com`. Это открытие способно с самого начала облегчить работу создателю CDF-файла. В этом случае следует добавить в элемент `CHANNEL` атрибут, определяющий базовый URL-адрес, из которого получаются все соответственные URL-адреса внутри CDF-файла. Подобное добавление атрибута не является обязательным, но намного облегчает создание CDF-файла, поскольку в дальнейшем не приходится полностью набирать все URL-адреса. Добавляемый атрибут – это атрибут `BASE` (Основа).

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http://www.xyzcorp.com/">

</CHANNEL>
```

Следующий атрибут, который можно добавить, – HREF. Этот атрибут определяет страницу, на которую в первую очередь направляется браузер IE4, когда подписчик открывает канал. Мы можем считать ее основной страницей канала. В нашем примере это страница INDEX.HTM.

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http://www.xyzcorp.com/" HREF="index.htm" >

</CHANNEL>
```

Существует еще три атрибута, которые можно добавить в элемент CHANNEL. Это – PRECACHE (Предварительное кэширование), LEVEL (Уровень) и LASTMOD (Последний режим). Для начала рассмотрим атрибут PRECACHE.

Атрибут PRECACHE указывает, следует ли загружать из Сети и размещать содержание канала в папке **Временные файлы Internet** (Temporary Internet Files) операционной системы Windows. Если атрибут PRECACHE определен как "YES" или опущен (что дает такой же эффект, как "YES", поскольку значение "YES" является значением по умолчанию), содержимое будет загружено только в том случае, если подписчик выбрал «загружать содержимое» (download the content) во время начальной процедуры подписки на канал.

Атрибут LEVEL используется совместно с атрибутом PRECACHE, чтобы указать, на сколько ссылок в глубину браузер IE4 выполнит *проход по ссылкам* (link crawl), чтобы кэшировать страницы. «Пройти по ссылкам» – это термин, используемый для описания свойства браузера IE4 автоматически следовать по каждой гиперссылке на странице и кэшировать все содержимое (HTML-документы, изображения, звуковые файлы и все содержимое фреймов, если ссылка указывает на набор фреймов) для просмотра в автономном режиме. Значение по умолчанию атрибута LEVEL равно "0". При данном значении кэшируется только URL, указанный в атрибуте HREF. Для атрибута LEVEL возможны следующие значения (рис. 13.6, 13.7, 3.8): "0", "1", "2" или "3".

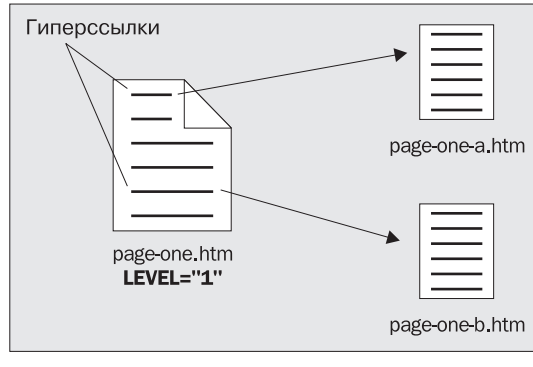


Рис. 13.6. Иллюстрация действия атрибута LEVEL. Значение LEVEL=1

Если установлено неверное значение атрибута LEVEL (например, "4"), то браузер IE4 пройдет по ссылкам вниз до третьего уровня и кэширует все встреченные страницы (и их содержание), нодальше не пойдет (рис. 13.9).

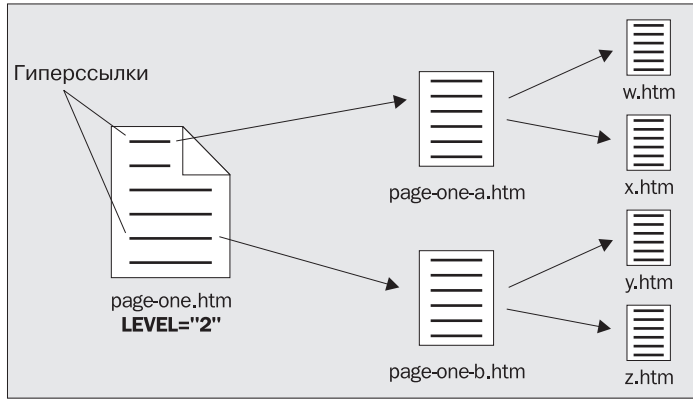


Рис. 13.7. Иллюстрация действия атибута LEVEL. Значение LEVEL=2

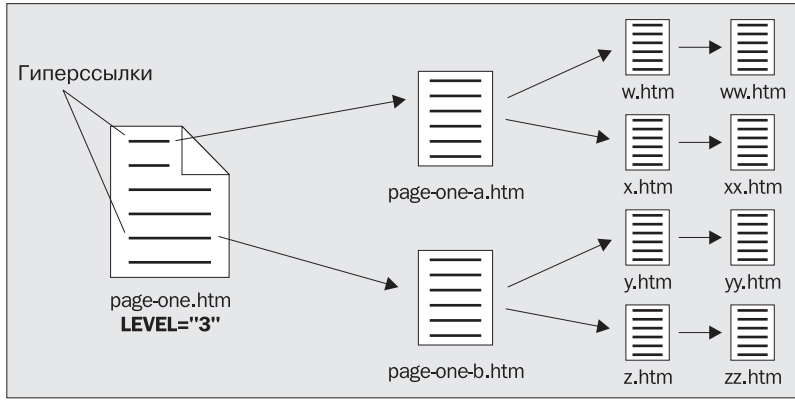


Рис. 13.8. Иллюстрация действия атибута LEVEL. Значение LEVEL=3

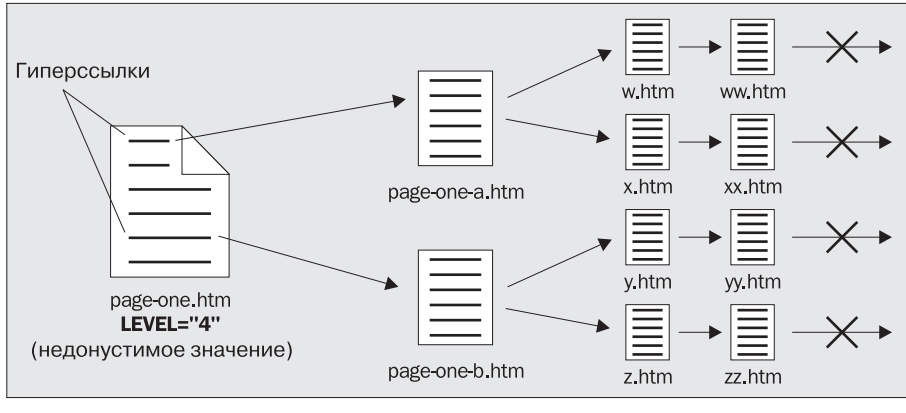


Рис. 13.9. Превышение допустимого значения (LEVEL=4)

Влияние атрибута PRECACHE на атрибут LEVEL

Если значение атрибута `PRECACHE` выставлено равным `"NO"`, то атрибут `LEVEL` игнорируется вне зависимостиот того, каково его значение.

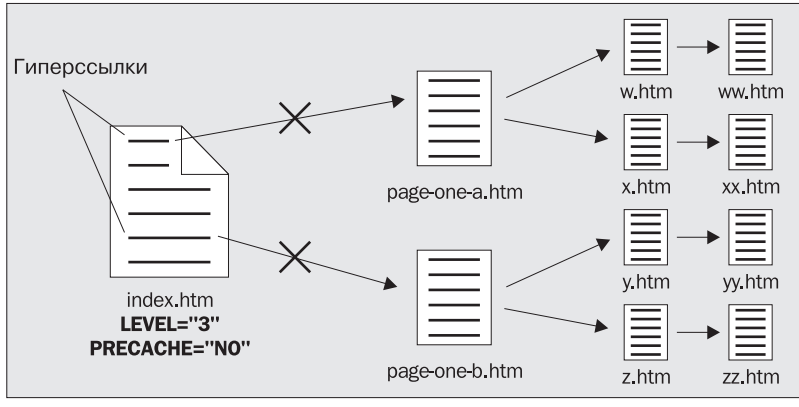


Рис. 13.10. Влияние атрибута PRECACHE на атрибут LEVEL. Независимо от значения атрибута LEVEL проход по ссылкам не происходит

Использование атрибута LASTMOD

Третий рассматриваемый нами атрибут – это атрибут `LASTMOD`. Этот атрибут определяет дату и время последнего обновления страницы (по Гринвичу, GMT), указанной атрибутом `HREF`. Атрибут `LASTMOD` имеет форму: `yyyy-mm-ddThh:mm` (табл. 13.3).

Таблица 13.3. Значения атрибута LASTMOD

yyyy	Указывает год
mm	Указывает месяц (01-12)
dd	Указывает день (01-31)
hh	Указывает час (00-23)
mm	Указывает минуты (00-59)

Атрибут `LASTMOD` позволяет браузеру IE4 узнавать, изменилось ли содержание страницы с момента ее последней загрузки. Повторная загрузка из Сети произойдет только в том случае, если дата, связанная с материалами, помещенными в кэш, более ранняя, чем значение атрибута `LASTMOD`, указанное в CDF-файле. При таком управлении процессом загрузки информации уменьшается нагрузка на сервер и время работы подписчиков в диалоговом режиме, поэтому использование данного атрибута рекомендуется.

Каковы были пожелания корпорации XYZ Corp относительно работы со страницами ее Web-сайта? Предполагалось, что страницы должны быть кэшированы, поэтому был добавлен атрибут `LASTMOD`, однако для корпорации нежелательно, чтобы осуществлялся переход по ссылкам на другие страницы. Далее приведены атрибуты, необходимые для этого.

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.htm" _PRECACHE="YES"
LEVEL="0" LASTMOD="1998-06-01T12:00">
</CHANNEL>
```

Атрибут LASTMOD указывает, что последний раз страница была обновлена в полдень 1-го июня 1998 года.

Аннотации и заголовки

Ранее упоминалось, что в канале могут определяться заголовки и аннотации. Названия и аннотации обычно применяются в особых случаях и по определенным целям, так что включать их следует только для полноты картины. Оба элемента TITLE (Заголовок) и ABSTRACT (Аннотация) являются дочерними элементами элемента CHANNEL. Элемент TITLE используется, чтобы определить заголовок канала по умолчанию, который появится в панели каналов (подписчик может изменить этот заголовок во время процесса подписки на канал). В то же время значение элемента ABSTRACT будет появляться в виде подсказки (ToolTip) при подведении курсора к кнопке канала в подокне каналов или на панели каналов (рис. 13.11).

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.htm" _PRECACHE="YES"
LEVEL="0" LASTMOD="1998-06-01T12:00">

<TITLE>Здесь расположен заголовок</TITLE>
<ABSTRACT>А здесь находится аннотация</ABSTRACT>

</CHANNEL>
```

Тестирование CDF-файла

Итак, мы создали CDF-файл, который будет работать, если корпорация XYZ Corp разместит его на сервере. В каком месте разместить CDF-файл, это прерогатива разработчика канала. Если для элемента CHANNEL был определен атрибут BASE, то CDF-файл можно просто поместить в место, указанное атрибутом BASE. Обеспечить доступ подписчиков к CDF-файлу можно при помощи обыкновенной гиперссылки. Единственное ограничение, налагаемое на местоположение CDF-файла, таково: CDF-файл должен быть помещен в тот же самый домен, в котором находится содержание канала, поскольку, по соображениям безопасности, браузер IE4 не имеет доступ к содержанию, находящемуся не в том же самом домене, что и CDF-файл.

Загрузка CDF-файла в браузер привела бы к загрузке и кэшированию основной титульной страницы канала и к его добавлению на панель каналов и в подокно каналов. Надстроим созданный CDF-файл, присоединив к нему другие страницы.

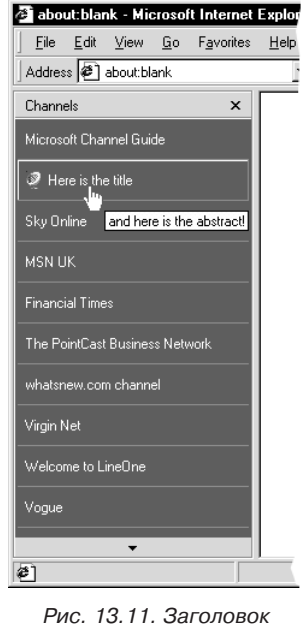


Рис. 13.11. Заголовок и аннотация канала

Присоединение содержания к CDF-файлу

Присоединение страниц к CDF-файлу требует помещения в элемент `CHANNEL` еще одного дочернего элемента, элемента `ITEM` (Пункт). Пункт – это единица информации, которая обычно соответствует Web-странице. В активном канале пункты появляются в навигационной иерархии, видимой в подокне каналов. При изучении элемента `CHANNEL` нам уже встречались четыре атрибута элемента `ITEM`. Этими атрибутами были атрибуты `HREF`, `LASTMOD`, `PRECACHE` и `LEVEL`.

Добавим в CDF-файл первый элемент `ITEM`. Для начала можно присоединить страницу каталога.

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.htm" _PRECACHE="YES"
LEVEL="0" LASTMOD="1998-06-01T12:00">

<TITLE>Здесь расположен заголовок</TITLE>
<ABSTRACT>А здесь находится аннотация</ABSTRACT>

<ITEM HREF="catalog.htm" PRECACHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">

</ITEM>

</CHANNEL>
```

Теперь в элемент `ITEM` добавим элементы `TITLE` и `ABSTRACT` так же, как они добавлялись в элемент `CHANNEL`.

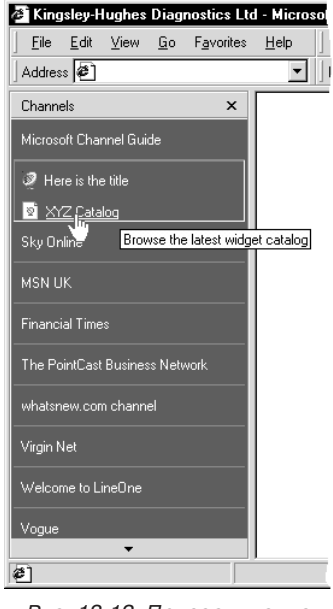


Рис. 13.12. Присоединение страницы к CDF-файлу

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.
htm" _PRECACHE="YES" LEVEL="0" LASTMOD="1998-06-
01T12:00">

<TITLE>Здесь расположен заголовок</TITLE>
<ABSTRACT>А здесь находится аннотация</ABSTRACT>

<ITEM HREF="catalog.htm" PRECACHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">
<TITLE>Каталог XYZ</TITLE>
<ABSTRACT>Просмотр новейшего каталога безделушек</
ABSTRACT>
</ITEM>

</CHANNEL>
```

Если загрузить CDF-файл на сервер и просмотреть результат (рис. 13.12), можно убедиться, что браузер IE4 автоматически добавил в иерархию первый пункт (`ITEM`).

Теперь необходимо присоединить страницу «Специальные предложения». Однако здесь есть небольшая хитрость. Не хотелось бы, чтобы эта страница была видна, как и каталог, в подокне каналов, лучше чтобы доступ к ней осуществлялся по гиперссылке со страницы каталога.

Как это можно осуществить? Поскольку на странице каталога есть несколько ссылок на страницы, которые не должны загружаться, автоматический переход по этим ссылкам был предотвращен выставлением значения атрибута `level` в CDF-файле равным `"0"`. Что же можно предпринять в данной ситуации? Прежде всего можно присоединить эту страницу к CDF-файлу таким же образом, каким была присоединена страница каталога (только на этот раз можно пропустить элементы `TITLE` и `ABSTRACT`, поскольку нет нужды в показе подобной информации в подокне каналов):

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.htm" _PRECACHE="YES"
LEVEL="0" LASTMOD="1998-06-01T12:00">

<TITLE>Здесь расположен заголовок</TITLE>
<ABSTRACT>А здесь находится аннотация</ABSTRACT>

<ITEM HREF="catalog.htm" PRECACHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">
<TITLE>Каталог XYZ</TITLE>
<ABSTRACT>Просмотр новейшего каталога безделушек</ABSTRACT>
</ITEM>

<ITEM HREF="special.htm" PRECACHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">
</ITEM>

</CHANNEL>
```

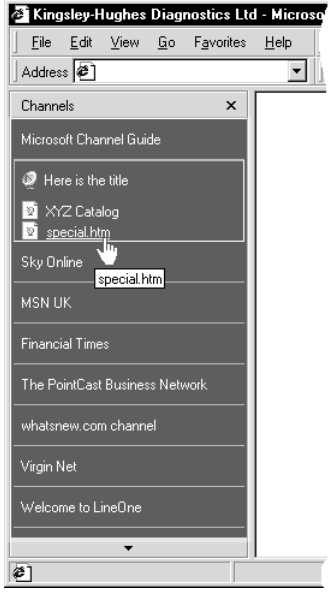


Рис. 13.13. Присоединение страницы special.htm к CDF-файлу

Если загрузить эту страницу в браузер, то видно (рис. 13.13), что только что присоединенная страница показана в подокне каналов.

Обратите внимание, поскольку элементы `TITLE` и `ABSTRACT` не были указаны, то в качестве значения обоих элементов используется имя файла. Как же спрятать этот пункт `ITEM`? Для этого потребуется новый дочерний элемент этого элемента `ITEM`. Этим дочерним элементом будет элемент `USAGE` (использование). Элемент `USAGE` может принимать шесть различных значений (табл. 13.4).

Для наших целей необходимо выставить значение элемента `USAGE` равным `"NONE"`.

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.
htm" _PRECACHE="YES" LEVEL="0" LASTMOD="1998-06-
01T12:00">

<TITLE>Здесь расположен заголовок</TITLE>
<ABSTRACT>А здесь находится аннотация</ABSTRACT>

<ITEM HREF="catalog.htm" PRECASHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">
<TITLE>Каталог XYZ</TITLE>
```

```
<ABSTRACT>Просмотр новейшего каталога безделушек</ABSTRACT>
</ITEM>

<ITEM HREF="special.htm" PRECACHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">
<USAGE VALUE="NONE"></USAGE>
</ITEM>

</CHANNEL>
```

Таблица 13.4. Возможные значения элемента USAGE

Значение	Описание
Channel (Канал)	Пункты (ITEM), имеющие элемент USAGE с таким значением, будут показаны в подокне браузера. Данное значение элемента USAGE является значением по умолчанию
Email (Электронная почта)	Один пункт, имеющий элемент USAGE с таким значением, посылается по электронной почте подписчику после того, как произошло обновление содержания канала (в CDF-файле можно указать только один элемент USAGE со значением равным "Email"). Доставка сообщения в почтовый ящик пользователя осуществляется браузером IE4
NONE (Ничто)	Если в элементе ITEM есть только один элемент USAGE со значением "NONE", то этот пункт не будет показан в подокне каналов
ScreenSaver (Заставка) (Или "SmartScreen" (Изящный экран) — торговая марка компании PointCast Inc.)	Один пункт, имеющий элемент USAGE с таким значением, будет использован в качестве заставки (в одном CDF-файле должна быть одна заставка)
DesktopComponent (Компонент рабочего стола)	Пункты, имеющие элементы USAGE с таким значением, будут показаны в рамке, расположенной на активном рабочем столе. Пункт CDF с таким значением элемента USAGE может быть использован только в контексте пункта активного рабочего стола. Активный канал требует отдельного CDF-файла
SoftwareUpdate (Обновление программного обеспечения)	Такое значение элемента USAGE указывает на то, что CDF-файл используется для канала распространения программного обеспечения (Software Distribution Channel), то есть для осуществления автоматического обновления программного обеспечения по Web. Для канала распространения программного обеспечения также требуется файл открытого описания программного обеспечения (Open Software Description – OSD)

Обратите внимание, что для этого элемента требуется закрывающий тэг `</USAGE>`. Чтобы доказать, что вставка нового элемента действует, загрузим CDF-файл в браузер IE4 (рис. 13.14).

Следующие две страницы, которые хотелось бы присоединить к CDF-файлу – это «Сроки и Условия» и «Гарантии». Можно было бы просто добавить страницы как два новых пункта, но можно поступить более «тонко». Хотелось бы ввести эти страницы в подканал в подокне каналов. Подканалы можно рассматривать, как папки Windows, внутри которых находятся файлы. Подканалы – прекрасный способ организовать схему расположения информации в подокне каналов и распределить темы по категориям, чтобы пользователям было легче

находить то, что им нужно. Так же, как вы не поместили бы все файлы на вашем компьютере в одну папку, потому что поиск в ней превратился бы в кошмар, не стоит поступать подобным образом и при организации навигационной иерархии в подокне каналов.

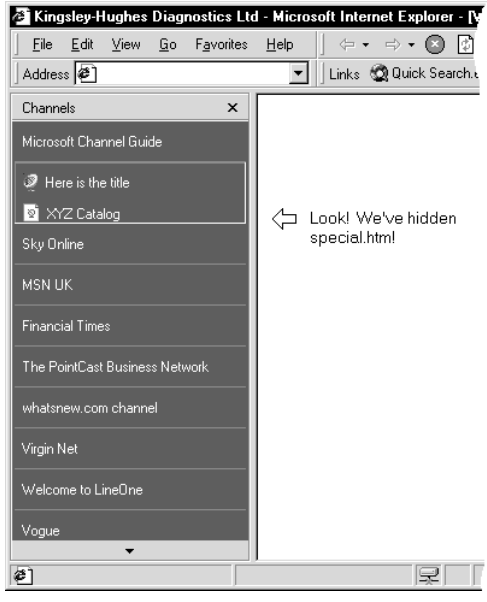


Рис. 13.14. Использование элемента USAGE

Учтите, что использование подканалов просто создает список «виртуальных» папок в подокне каналов и никоим образом не отражает схему размещения содержимого на вашем сервере. Подканалы образуются путем вложения элементов `CHANNEL` в первоначальный элемент `CHANNEL`. Начнем со следующего:

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.htm" _PRECACHE="YES"
LEVEL="0" LASTMOD="1998-06-01T12:00">

<TITLE>Здесь расположен заголовок</TITLE>
<ABSTRACT>А здесь находится аннотация</ABSTRACT>

<ITEM HREF="catalog.htm" PRECACHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">
<TITLE>Каталог XYZ</TITLE>
<ABSTRACT>Просмотр новейшего каталога безделушек</ABSTRACT>
</ITEM>

<ITEM HREF="special.htm" PRECACHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">
<USAGE VALUE="NONE"></USAGE>
</ITEM>

<CHANNEL>
```

```
</CHANNEL>
</CHANNEL>
```

Сейчас, чтобы выделить эту «папку» подканала в подокне каналов, можно дать ей собственный заголовок (TITLE) и аннотацию (ABSTRACT).

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.htm" PRECACHE="YES"
LEVEL="0" LASTMOD="1998-06-01T12:00">
.
...
.
<ITEM HREF="special.htm" PRECACHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">
<USAGE VALUE="NONE"></USAGE>
</ITEM>

<CHANNEL>
<TITLE>МАЛАЯ ПЕЧАТЬ!</TITLE>
<ABSTRACT>Закажите МАЛУЮ ПЕЧАТЬ XYZ Corp!</ABSTRACT>
</CHANNEL>
</CHANNEL>
```

И наконец, можно добавить в подканал элементы ITEM:

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.htm" _PRECACHE="YES"
LEVEL="0" LASTMOD="1998-06-01T12:00">

...

<CHANNEL>
<TITLE>МАЛАЯ ПЕЧАТЬ!</TITLE>
<ABSTRACT>Закажите МАЛУЮ ПЕЧАТЬ XYZ Corp!</ABSTRACT>
<ITEM HREF="terms.htm" PRECACHE="YES" LEVEL="0" LASTMOD="1998-06-01T12:00">
<TITLE>Сроки и условия XYZ Corp </TITLE>
<ABSTRACT>Вся малая печать по срокам и условиям</ABSTRACT>
</ITEM>
<ITEM HREF="warranty.htm" PRECACHE="YES" LEVEL="0" LASTMOD="1998-06-01T12:00">
<TITLE>Гарантии XYZ Corp </TITLE>
<ABSTRACT>Гарантийные условия по нашим безделушкам</ABSTRACT>
</ITEM>
</CHANNEL>
</CHANNEL>
```

Кодировка символов для CDF

Обратите внимание, что название «Сроки и Условия» (Terms and Conditions) было написано именно через «и», а не «Сроки & Условия» (Terms & Conditions). Как и в XML, в CDF-файлах некоторые символы должны быть закодированы, если их помещают между открывающим и закрывающим тэгами элемента. Эти символы перечислены в табл. 13.5.

Таблица 13.5. Кодируемые символы для CDF-файлов

Символ	Закодированное значение
< (меньше чем)	<
> (больше чем)	>
' (апостроф)	'
" (кавычки)	"
& (амперсанд)	&

Если нужно вставить текст в виде «Сроки & Условия» (Terms & Conditions), можно написать:

```
Сроки &amp; Условия
```

При несоблюдении этих правил во время подписки на канал или в процессе обновления канала возникнут ошибки разбора.

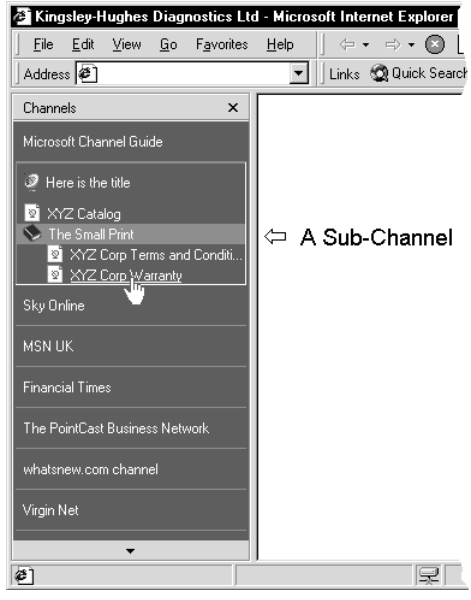


Рис. 13.15. Подканал в CDF-файле

Дальнейшее подсоединение страниц к CDF-файлу

Теперь можно без затруднений добавить оставшиеся две страницы в другой подканал, который мы хотим создать в окне каналов.

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.htm" _PRECACHE="YES"
LEVEL="0" LASTMOD="1998-06-01T12:00">

<TITLE>Здесь расположен заголовок</TITLE>
<ABSTRACT>А здесь находится аннотация</ABSTRACT>
.
<CHANNEL>
<TITLE>МАЛАЯ ПЕЧАТЬ!</TITLE>
<ABSTRACT>Закажите МАЛУЮ ПЕЧАТЬ XYZ Corp!</ABSTRACT>
<ITEM HREF="terms.htm" PRECACHE="YES" LEVEL="0" LASTMOD="1998-06-01T12:00">
<TITLE>Сроки и условия XYZ Corp </TITLE>
<ABSTRACT>Вся малая печать по срокам и условиям</ABSTRACT>
</ITEM>
<ITEM HREF="warranty.htm" PRECACHE="YES" LEVEL="0" LASTMOD="1998-06-01T12:00">
<TITLE>Гарантии XYZ Corp </TITLE>
<ABSTRACT>Гарантийные условия по нашим безделушкам</ABSTRACT>
</ITEM>
</CHANNEL>

<CHANNEL>
<TITLE>В XYZ Corp</TITLE>
<ABSTRACT>Как дела в XYZ Corp</ABSTRACT>
```

```
<ITEM HREF="new.htm" PRECACHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">
<TITLE>Что нового в XYZ Corp</TITLE>
<ABSTRACT>Узнайте, что нового в XYZ Corp</ABSTRACT>
</ITEM>
<ITEM HREF="happening.htm" PRECACHE="YES" LEVEL="0" LASTMOD="1998-06-01T12:00">
<TITLE>Что происходит в XYZ Corp</TITLE>
<ABSTRACT>Держитесь в курсе того, что происходит в XYZ Corp</ABSTRACT>
</ITEM>
</CHANNEL>

</CHANNEL>
```

Теперь у нас имеется полностью работающий CDF-файл. Корпорация XYZ Corp может положить его на свой сервер, добавить соответствующую ссылку на него и позволить клиентам и покупателям подписаться на свой новый активный канал с торговой маркой корпорации. (Быть может, корпорации понадобится привести в порядок названия (`TITLE`) и аннотации (`ABSTRACT`), которые были помещены в ее канал!). Однако все сделанное к этому моменту – еще только самое начало. XML в виде CDF можно использовать в браузере IE4 для достижения еще большего эффекта. Например, можно:

- добавить *активную заставку* (Active Screensaver) в содержимое канала;
- добавить расписание (`SCHEDULE`) для автоматизации процесса обновления;
- добавить логотипы и пиктограммы на панель каналов и подокно каналов, чтобы индивидуализировать канал.

Изучим эти эффекты по очереди, начиная с использования активной заставки.

Добавление активной заставки

Активная заставка (Active Screensaver) — это новая концепция, ставшая возможной благодаря браузеру IE4. Легко устанавливаемая, активная заставка представляет собой обыкновенную HTML-страницу, которая показывается браузером IE4 как заставка. Любой пользователь, знающий HTML (или еще лучше DHTML, потому что именно DHTML делает заставку «активной») и способный написать CDF-файл, может ее создать.

Предположим, что кто-то в корпорации XYZ Corp подготовил прекрасную DHTML-заставку, и корпорация хочет включить ее вместе со всем содержанием в канал. Заставка, названная `ssaver.htm`, была размещена в корне Web-сайта корпорации XYZ Corp. Необходимо решить: как присоединить заставку к CDF-файлу? Вернемся к тому месту, где происходило знакомство с элементом `USAGE`, являющимся дочерним элементом элемента ITEM. Одним из возможных значений этого дочернего элемента было значение `"ScreenSaver"`.

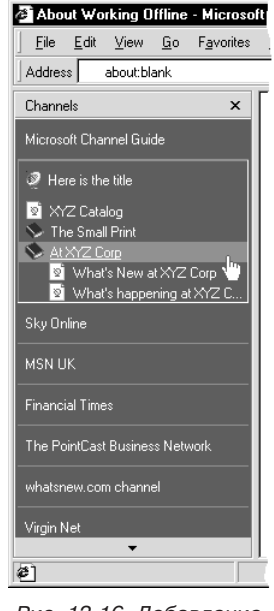


Рис. 13.16. Добавление двух страниц в новый подканал

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.htm" _PRECACHE="YES"
LEVEL="0" LASTMOD="1998-06-01T12:00">
.
<ITEM HREF="special.htm" PRECACHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">
<USAGE VALUE="NONE"></USAGE>
</ITEM>
<ITEM HREF="ssaver.htm" PRECACHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">
<USAGE VALUE="ScreenSaver"></USAGE>
</ITEM>
<CHANNEL>
<TITLE>МАЛАЯ ПЕЧАТЬ!</TITLE>
<ABSTRACT>Закажите МАЛУЮ ПЕЧАТЬ XYZ Corp!</ABSTRACT>
<ITEM HREF="terms.htm" PRECACHE="YES" LEVEL="0" LASTMOD="1998-06-01T12:00">
<TITLE>Сроки и условия XYZ Corp </TITLE>
<ABSTRACT>Вся малая печать по срокам и условиям</ABSTRACT>
</ITEM>
.
</CHANNEL>
</CHANNEL>
```

В настоящий момент существует возможность присоединения только одной активной заставки к одному CDF-файлу. Если в CDF-файле указано более одной заставки, то использоваться будет только первая.

Как только загрузка заставки на компьютер подписчика закончится, перед ним появится диалоговое окно, изображенное на рис. 13.17 (если только ранее, при первых появлениях данного окна, он не выбралопцию **Don't ask me again** (Больше не задавать этот вопрос)).

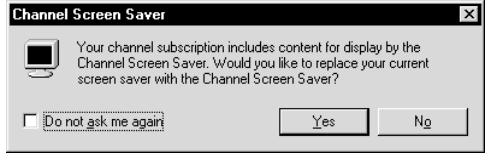


Рис. 13.17. Окно с предложением заменить обычную заставку на заставку канала

Если у подписчика уже установлена другая активная заставка, то новая и старая заставки будут показываться на экране поочередно через определенные интервалы времени (по умолчанию через 30 секунд). Не существует ограничений на число активных заставок, установленных одновременно на компьютере пользователя.

Автоматизация обновления канала

Одна из мощных возможностей, доступных теперь разработчику канала, – это возможность осуществлять регулярную *принудительную доставку* (push) обновленного содержимого канала зарегистрированным подписчикам. Не следует думать, что слова «принудительная доставка» подразумевают необходимость разработки сложного сервера или запуска новых расширений сервера. На самом

деле термин «принудительная доставка» несколько искажает смысл этого способа, здесь более подходит термин «управляемое получение» (managed pull), поскольку вся работа (за исключением создания CDF-файла) выполняется на стороне клиента браузером IE4 (в настоящее время единственный реальный пример настоящей «принудительной доставки» (push) — это электронная почта (e-mail)!). Разработчику канала необходимо просто указать браузеру, что он хочет доставлять обновленное содержание. Осуществить это можно, используя XML в CDF-файле.

Сообщить браузеру IE4, что будет осуществляться принудительная доставка, можно при помощи элемента `SCHEDULE` (Расписание), который является дочерним элементом элемента `CHANNEL`.

Для начала можно добавить этот элемент в ранее нами созданный CDF-файл.

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.htm" _PRECACHE="YES"
LEVEL="0" LASTMOD="1998-06-01T12:00">

<TITLE>Активный канал XYZ Corp</TITLE>
<ABSTRACT>Добро пожаловать на активный канал XYZ Corp - Дом Безделушек</ABSTRACT>

<SCHEDULE>
...
</SCHEDULE>

<ITEM HREF="catalog.htm" PRECACHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">
<TITLE>Каталог XYZ</TITLE>
<ABSTRACT>Просмотр новейшего каталога безделушек</ABSTRACT>
</ITEM>

<CHANNEL>
...
</CHANNEL>

</CHANNEL>
```

В одном CDF-файле можно указать только одно расписание (`SCHEDULE`), и элемент `SCHEDULE` должен быть вставлен в канал верхнего уровня.

В начале главы упоминалось, что корпорация XYZ Corp обновляет свой каталог еженедельно по пятницам. Хотелось бы отразить это обстоятельство в создаваемом расписании. Для этого используется пустой дочерний элемент, называемый `INTERVALTIME` (Временной интервал), элемента `SCHEDULE`.

Элемент `INTERVALTIME` указывает, как часто должна производиться попытка обновления канала. Если есть желание еженедельно получать на канале свежие данные, значение элемента `INTERVALTIME` можно выставить равным `DAY="7"` (День). Другие возможные варианты: `HOUR` (Час) и `MIN` (Минута).

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.htm" PRECACHE="YES"
LEVEL="0" LASTMOD="1998-06-01T12:00">

<TITLE>Активный канал XYZ Corp</TITLE>
```

```
<ABSTRACT>Добро пожаловать на активный канал XYZ Corp - Дом Безделушек</ABSTRACT>

<SCHEDULE>
<INTERVALTIME DAY="7"/>
</SCHEDULE>

...
```

Обратите внимание, что этому элементу не нужен закрывающий тэг, поскольку он является пустым элементом, но косая черта в закрывающем ограничителе необходима.

Таким образом, часть наших желаний исполнилась: канал будет обновляться еженедельно. Однако здесь мы столкнулись с небольшой загвоздкой: у нас нет рычагов управления началом цикла обновления. При использовании только что созданного CDF-файла, обновление информации у пользователя, подписавшегося на канал в понедельник, должно произойти в следующий понедельник, если только не будет проблем с подключением к сети. Аналогично, если пользователь подписался в четверг, то у него следующее обновление содержания канала должно произойти тоже в четверг. Это означает, что каталог устареет еще до того, как подписчики его получат. Необходимо средство управления датой, с которой начнется выполнение расписания. Атрибут **STARTDATE** (Дата запуска) элемента **SCHEDULE** – это как раз то, что нужно.

Если необходимо, чтобы начало выполнения расписания пришлось на пятницу, необходимо проделать небольшой трюк: выбрать некоторый пятничный день в прошлом и добавить дату этого дня в элемент **SCHEDULE**.

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.htm" PRECACHE="YES"
LEVEL="0" LASTMOD="1998-06-01T12:00">

<TITLE>Активный канал XYZ Corp</TITLE>
<ABSTRACT>Добро пожаловать на активный канал XYZ Corp - Дом Безделушек</ABSTRACT>

<SCHEDULE STARTDATE="1998-05-29">
<INTERVALTIME DAY="7"/>
</SCHEDULE>

...
```

Мы установили день, с которого начинается выполнение расписание, теперь хотелось бы, чтобы процесс обновления происходил как можно ближе ко дню обновления каталога. Для этого необходимо обратиться к другому пустому дочернему элементу элемента **SCHEDULE**: к элементу **LATESTTIME** (Самое позднее). Этот элемент указывает самый поздний срок (в указанном интервале обновлений), до которого должно произойти обновление. Если каталог обновляется в пятницу днем, то можно позволить подписчикам проводить обновление содержания на их компьютерах до вторника. В этом случае следует добавить одну строчку в CDF-файл.

```
<?xml version="1.0" encoding="UTF-8"?>
    <CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.htm" PRECACHE="YES"
                    LEVEL="0" LASTMOD="1998-06-01T12:00">
<TITLE>Активный канал XYZ Corp</TITLE>
```

```
<ABSTRACT>Добро пожаловать на активный канал XYZ Corp - Дом Безделушек</ABSTRACT>

<SCHEDULE STARTDATE="1998-05-29">
<INTERVALTIME DAY="7"/>
<LATESTTIME DAY="4"/>
</SCHEDULE>

...
```

Изменение записи `<LATESTTIME DAY="4" />` на `<LATESTTIME DAY="2" />` означало бы, что подписчики могут проводить обновление до воскресенья. Если браузеру IE4 не удастся обновить содержание до указанного срока, то более никаких попыток осуществить обновление предприниматься не будет, вплоть до начала следующего цикла.

В настоящий момент благодаря грамотно установленному расписанию CDF-файла подписчики получают самую свежую информацию; при этом уменьшена вероятность того, что они работают со старыми данными.

Сейчас, когда использован элемент `SCHEDULE`, стала ясна причина применения атрибута `LASTMOD`. С его помощью удается избежать ситуации, когда пользователи загружают информацию, которая у них уже есть, поскольку при наличии атрибута `LASTMOD` загрузка информации происходит только в том случае, если дата, указанная атрибутом `LASTMOD`, более поздняя, чем дата материалов, помещенных в кэш. Такая настройка процесса обновления сокращает время загрузки и плату за работу в Сети, повышает производительность системы. Это снижает и нагрузку на сервер, на котором содержится сайт; а попытка добиться этого никогда не считалась лишней. Однако не забывайте обновлять даты, указанные атрибутом `LASTMOD` для содержания канала, иначе подписчики никогда не увидят ваше новое содержание!

Индивидуальное оформление канала

Как много компаний в прошлом мечтали создать в среде Windows ссылку на свой Web-сайт, расположенную на рабочем столе своих клиентов и покупателей. А если бы они могли оформить эту ссылку специальными логотипами и пиктограммами? Безусловно, это было бы столь заманчиво, что вряд ли кто-нибудь упустил такую возможность. Ну что ж, если вам была нужна еще одна причина, по которой компаниям и организациям следует иметь активный канал, так это она и есть. У разработчика канала имеется четыре различных типа изображения для оформления канала (рис. 13.18):

- логотип для отображения на панели каналов (высота 32 пиксела, ширина 80 пикселов);
- логотип для отображения в подокне каналов (высота 32 пиксела, ширина 194 пиксела);
- пиктограмма для пунктов на панели каналов (высота 16 пикселов, ширина 16 пикселов);
- пктограмма для подканала (размер, как и у предыдущей пиктограммы).

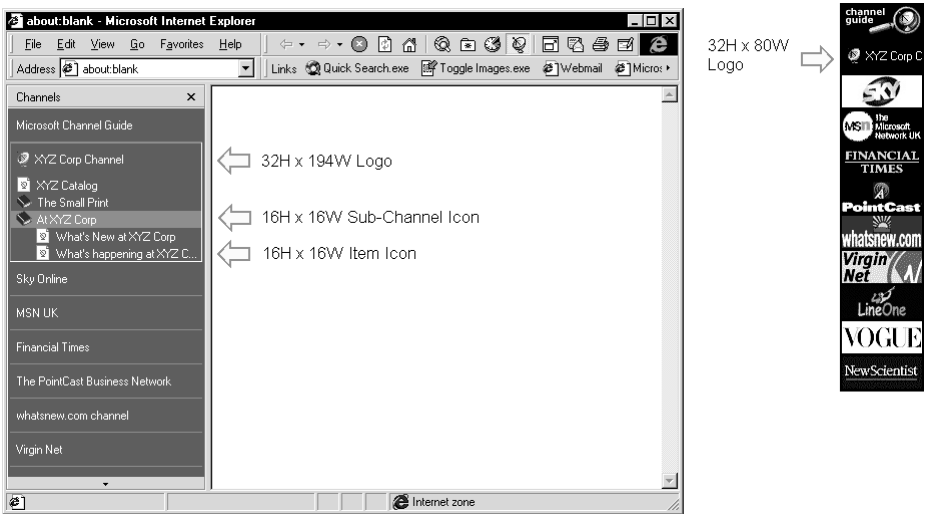


Рис. 13.18. индивидуальное оформление канала

Рассмотрим более внимательно, как можно добавить эти изображения в канал. Для начала рассмотрим логотипы для панели управления и подокна каналов. Чтобы добавить эти логотипы, используются, что не удивительно, пустые элементы, называемые `LOGO` (Логотип). Элемент `LOGO` имеет два атрибута: `HREF` и `STYLE` (Стиль). Атрибут `HREF` определяет URL-адрес изображения, которое будет использовано (можно использовать изображения в формате `GIF`, `JPG/JPEG` и в других форматах файлов, поддерживаемых браузером IE4; `GIF`-файлы с анимацией не поддерживаются). Атрибут `STYLE` указывает, каким образом должно быть использовано изображение; этот атрибут может принимать одно из трех значений (табл. 13.6).

Таблица 13.6. Значения атрибута STYLE элемента LOGO

Значение	Описание
IMAGE	Логотип панели каналов
IMAGE-WIDE	Логотип подокна каналов
ICON	Пиктограмма для пунктов в подокне каналов или для подканалов
	Каждый элемент ITEM может иметь свою пиктограмму (тэг LOGO в этом случае становится дочерним элементом элемента ITEM) или все элементы ITEM могут иметь одну одинаковую пиктограмму (элемент LOGO в этом случае становится дочерним элементом элемента CHANNEL)

Вставим элемент `LOGO` в CDF-файл, чтобы добавить логотипы панели каналов и подокна каналов.

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.htm" _PRECACHE="YES"
LEVEL="0" LASTMOD="1998-06-01T12:00">

<TITLE>Активный канал XYZ Corp</TITLE>
```

```
<ABSTRACT>Добро пожаловать на активный канал XYZ Corp - Дом Безделушек</ABSTRACT>
<LOGO HREF="button.gif" STYLE="IMAGE"/>
<LOGO HREF="button_w.gif" STYLE="IMAGE-WIDE"/>
<SCHEDULE STARTDATE="1998-05-29">
<INTERVALTIME DAY="7"/>
<LATESTTIME="4"/>
</SCHEDULE>
...
```

Рис. 13.19. Логотип, вставленный в подокно каналов браузера

Эти две строчки добавляют логотипы в подокно каналов (рис. 13.19) и на панель каналов (рис. 13.20).

Теперь можно закончить оформление канала, добавив пиктограммы для элементов **ITEM** и для подканалов. Пиктограммы для подканалов добавляются следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.
htm" PRECACHE="YES" LEVEL="0" LASTMOD="1998-06-
01T12:00">
...
<ITEM HREF="ssaver.htm" PRECACHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">
<USAGE VALUE="ScreenSaver"></USAGE>
</ITEM>
<CHANNEL>
<LOGO HREF="http://www.xyzcorp.com /icon_s/gif"
STYLE="ICON"/>
<TITLE>МАЛАЯ ПЕЧАТЬ!</TITLE>
<ABSTRACT>Закажите МАЛУЮ ПЕЧАТЬ XYZ Corp!</ABSTRACT>
<ITEM HREF="terms.htm" PRECACHE="YES" LEVEL="0" LASTMOD="1998-06-01T12:00">
<TITLE>Сроки и условия XYZ Corp </TITLE>
<ABSTRACT>Вся малая печать по срокам и условиям</ABSTRACT>
</ITEM>
...
<CHANNEL>
<LOGO HREF="http://www.xyzcorp.com /icon_s/gif" STYLE="ICON"/>
<TITLE>В XYZ Corp</TITLE>
<ABSTRACT>Как дела в XYZ Corp</ABSTRACT>
<ITEM HREF="new.htm" PRECACHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">
<TITLE>Что нового в XYZ Corp</TITLE>
<ABSTRACT>Узнайте, что нового в XYZ Corp</ABSTRACT>
</ITEM>
<ITEM HREF="happening.htm" PRECACHE="YES" LEVEL="0" LASTMOD="1998-06-01T12:00">
<TITLE>Что происходит в XYZ Corp</TITLE>
<ABSTRACT>Держитесь в курсе того, что происходит в XYZ Corp</ABSTRACT>
</ITEM>
...
```

```
</CHANNEL>
</CHANNEL>
```

А теперь можно добавить в CDF-файл пиктограммы для элементов **ITEM**:

```
<?xml version="1.0" encoding="UTF-8"?>
<CHANNEL BASE ="http:// www.xyzcorp.com/" HREF="index.htm"
PRECACHE="YES" LEVEL="0" LASTMOD="1998-06-01T12:00">
...
<ITEM HREF="catalog.htm" PRECACHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">
<LOGO HREF="http://www.xyzcorp.com /icon/gif" STYLE="ICON"/>
<TITLE>Каталог XYZ</TITLE>
<ABSTRACT>Просмотр новейшего каталога безделушек</ABSTRACT>
</ITEM>
...
<CHANNEL>
<LOGO HREF="http://www.xyzcorp.com /icon_s/gif"
STYLE="ICON"/>
<TITLE>МАЛАЯ ПЕЧАТЬ!</TITLE>
<ABSTRACT>Закажите МАЛУЮ ПЕЧАТЬ XYZ Corp!</ABSTRACT>
<ITEM HREF="terms.htm" PRECACHE="YES" LEVEL="0"
LASTMOD="1998-06-01T12:00">
<LOGO HREF="http://www.xyzcorp.com /icon/gif" STYLE="ICON"/>
<TITLE>Сроки и условия XYZ Corp </TITLE>
<ABSTRACT>Вся малая печать по срокам и условиям</ABSTRACT>
</ITEM>
<ITEM HREF="warranty.htm" PRECACHE="YES" LEVEL="0"
LASTMOD="1998-06-01T12:00">
<LOGO HREF="http://www.xyzcorp.com /icon/gif" STYLE="ICON"/>
<TITLE>Гарантии XYZ Corp </TITLE>
<ABSTRACT>Гарантийные условия по нашим безделушкам</ABSTRACT>
</ITEM>
</CHANNEL>
```

Рис. 13.20. Логотип, вставленный на панель каналов

```
<CHANNEL>
<LOGO HREF="http://www.xyzcorp.com /icon_s/gif" STYLE="ICON"/>
<TITLE>В XYZ Corp</TITLE>
<ABSTRACT>Как дела в XYZ Corp</ABSTRACT>
<ITEM HREF="new.htm" PRECACHE="YES" LEVEL="0"
_LASTMOD="1998-06-01T12:00">
<LOGO HREF="http://www.xyzcorp.com /icon/gif" STYLE="ICON"/>
<TITLE>Что нового в XYZ Corp</TITLE>
<ABSTRACT>Узнайте, что нового в XYZ Corp</ABSTRACT>
</ITEM>
<ITEM HREF="happening.htm" PRECACHE="YES" LEVEL="0" LASTMOD="1998-06-01T12:00">
<LOGO HREF="http://www.xyzcorp.com /icon/gif" STYLE="ICON"/>
<TITLE>Что происходит в XYZ Corp</TITLE>
<ABSTRACT>Держитесь в курсе того, что происходит в XYZ Corp</ABSTRACT>
```

```
</ITEM>
</CHANNEL>

</CHANNEL>
```

Все сделано! Окончательный результат представлен на рис. 13.21.

Рис. 13.21. Вставлены пиктограммы для пунктов в подокне каналов

Теперь сотрудникам корпорации XYZ Corp остается только сидеть и ждать, когда подписчики начнут толпами стекаться к каналу корпорации.

Заключение

В этом учебном примере мы рассмотрели основы построения каналов. Мы не изучали все элементы и атрибуты, а также их общий вклад в рабочий процесс канала. Эти вопросы находятся за рамками нашего учебного примера. В данной главе говорилось о том, как при помощи XML в форме CDF-файла улучшить функциональность существующего Web-сайта, усовершенствовать навигацию по узлу, облегчить конечным пользователям процесс обновления информации путем создания активного канала.

Почему компания Microsoft выбрала XML в качестве языка для CDF? Безусловно, можно было бы создать новый формат (как поступили разработчики Netscape со своим форматом, использующим JavaScript). Одна из причин заключается в том, что язык XML уже стал достаточно популярным в среде разработчиков. Другой неоспоримой причиной является то, что синтаксис языка XML очень прямолинеен и легко изучается; это ускоряет понимание новой технологии разработчиками Web.

Мы увидели, как следует использовать XML, чтобы реализовать новые возможности браузера IE4. Мы можем:

- создать структурный CDF-файл и добавить в него основные компоненты, составляющие канал;
- управлять кэшированием канала для последующего просмотра;
- создавать схему навигационной иерархии в подокне каналов и управлять ею;
- добавлять заставки в содержание канала;
- управлять процессом обновления канала, используя расписание (`SCHEDULE`);
- оформлять канал персональными логотипами и пиктограммами, появляющимися в подокне каналов и на панели каналов.

Мы также рассмотрели, как можно избежать некоторых типичных проблем, досаждающих разработчикам каналов. Удачного построения каналов!

Приложение А
Языки и обозначения

Данное приложение объясняет форму синтаксиса EBNF, используемую в различных спецификациях. Общее понимание синтаксиса, конечно, необходимо, однако углубленные знания (или работа с источником, подобным данному) понадобятся только в случае детального изучения спецификаций.

Запись в расширенной форме Бэкуса–Наура

Формальная грамматика языка XML дана с использованием записи в простой ***расширенной форме Бэкуса-Наура*** (Extended Backus-Naur Form, EBNF). Форма EBNF в основном предназначена для чтения машинами, но при разработке также предусматривалась ее удобочитаемость для человека. (Я полагаю, что это зависит от того, что подразумевать под «удобочитаемостью для человека».)

Запись EBNF или ее модифицированная форма используется в нескольких других спецификациях консорциума W3C; она также получила широкое распространение по всему компьютерному миру. Модифицированная форма применяется в DTD-определениях. Термины и их определения выделены в рамках, а затем даны детальные примеры их использования.

Определение #XN

`#XN` – представляет собой шестнадцатеричное целое `N`, как определено в ISO/IEC 10646.

Например, пробельные литеры в XML-документе определены следующим образом:

```
White Space
_ S ::= (#x20 |_#x9 |_#xD |_#xA)+
```

Это означает, что пробельные литеры представляют собой комбинацию одного или более (это обозначено символом `+`) символов ISO/IEC 10646 в любом порядке (это обозначено символом `|`):

- шестнадцатеричное число 20 (=ASCII десятичное 32, пробел);
- шестнадцатеричное число 9 (=ASCII десятичное 9, табуляция);
- шестнадцатеричное число D (=ASCII десятичное 13, новая строка);
- шестнадцатеричное число A (=ASCII десятичное 10, перевод строки).

Определение [A-zA-Z], [#xN-#xN]

`[A-zA-Z]`, `[#xN-#xN]` – соответствует любому символу в указанном диапазоне (диапазонах), включая границы.

Например, в спецификации XML разрешенный символ (char) определен следующим образом:

```
Char ::= #x9 |_#xA |_#xD |_[#x20-#xD7FF] |_[#xE000-#xFFFD] |_[#x10000-#x10FFFF]
```

`[#x20-#xD7FF]` означает, что любой символ из диапазона может быть использован как допустимый символ. (Сюда входят все ASCII-символы.) Применение `#x9`, `#xD`, `#xA` означает, что пробельные литеры могут быть включены в число разрешенных символов.

Еще раз обратите внимание на символ вертикальной черты (`|`), который устанавливает альтернативность.

Определение [^abc], [^#xN#xN#xN]

`[^abc]`, `[^#xN#xN#xN]` – символ `^` означает «исключая»:

```
Attvalue ::= '"' ([^<&"] |_Reference)* '"' |_ "'" ([^<&"] |_Reference)* "'"
```

В приведенной выше ссылке для `AttValue` выражение `[^<&]` означает, что ***ссылка*** (reference) – определенная где-то в другом месте спецификации – может принимать любое значение до тех пор, пока в нем не содержится символов `&` или `<`.

Определение [^a-z] [^#xN-#xN]

`[^a-z] [^#xN-#xN]` – это исключение, относящееся к диапазону значений.

Я не сумел найти образец в спецификации XML, но это может выглядеть примерно так:

```
FirstHalfAlphabet::=([^n-z] | alphabet)
```

Это означает, что `FirstHalfAlphabet` (Первая половина алфавита) может быть любой буквой алфавита (по-видимому, определенного где-то еще, как символы a-z) за исключением символов из диапазона n-z.

Определение "string" и 'string'

`"string"` и `'string'` – оба выражения относятся к буквенной строке.

В XML-спецификации есть следующее:

```
DefaultDecl::= '#REQUIRED' |_'#IMPLIED'
```

Это означает, что ***определение по умолчанию*** (default declaration) может принимать значение одной из двух строк: `'#REQUIRED'` (обязательное) или `'#DEFAULT'` (по умолчанию).

Определение (expression)

`(expression)` рассматривается как некая целая единица. Выражения (`expression`) могут быть сгруппированы, как это описано в данном перечне. Такая форма обозначения очень распространена во всех спецификациях консорциума W3C.

Определение A?

`A?` соответствует A или ничему; необязательному A.

Это означает, что включение `A` необязательно. В спецификации XML есть следующее определение декларации обычного объекта (general entity), где `S` обозначает пробельные литеры.

```
GEDecl::= '<!ENTITY' S Name S EntityDef S? '>'
```

Две первых пробельных литеры обязательны, последняя необязательна.

Определение A B

`A B` соответствует `A`, за которым следует `B`.

Например, в следующем определении общедоступного (public) ID-номера (идентификационный номер):

```
PublicID::='PUBLIC' S PubidLiteral
```

сообщается, что номер `PublicID` должен иметь форму строки `'PUBLIC'`, за которой следует пробельная литера, а за ней выражение `PubidLiteral` (которое определено в другом месте спецификации).

Определение A | B

`A | B` соответствует `A` или `B`, но не обоим вместе.

Выше мы уже встречали примеры таких выражений, но напомним: в следующем определении выражения `PEDed` его значением может быть `EntityValue` или `_ExternalID`, но не оба.

```
PEDed::= EntityValue |_ExternalID
```

Определение A–B

`A-B` – это любая строка, соответствующая `A`, но не соответствующая `B`.

В определении XML дан следующий пример:

```
PITarget::= Name-(('X' |_'x') ('M'|_'m') ('L'|_'l'))
```

Это означает, что `PITarget` может иметь любое значение имени (`Name`), определенное где-то в другом месте, исключая последовательность XML-элементов или любую комбинацию из этих букв, строчных или заглавных.

Определение A+

`A+` соответствует повторению `A` один или более раз:

```
CharRef::= '&#' [0-9]+ ';' |_'&#x' [0-9a-fA-F]+ ';'
```

Это означает, что величина `CharRef` должна начинаться со строки `'&#'`, за которой будет следовать либо комбинация из одной или нескольких цифр (от 1 до 9), либо комбинация из одной или нескольких цифр или букв от A до F, а завершаться величина будет точкой с запятой.

Определение A*

`A*` соответствует повторению `A` ноль или более раз:

```
EncName::=[A-Za-z] ([A-Za-z0-9._] |_'-')*
```

Это означает, что величина `EncName` должна состоять из одного символа, принадлежащего диапазону a-z (в верхнем или нижнем регистре) плюс некоторое количество необязательных дополнительных символов, которые могут принадлежать диапазону a-z, 0-9 (цифры), быть точкой, символом подчеркивания или дефисом.

Спецификация XML

Спецификация XML использует форму синтаксиса EBNF, описанную выше. Спецификация XML 1.0 может быть найдена по адресу:

http://www.w3.org/TR/REC-xml

Спецификация CSS1

Спецификация CSS1 использует модифицированный (и более доступный для человека) синтаксис EBNS (табл. A1). С ней можно ознакомиться по адресу:

http://www.w3.org/TR/REC- CSS1

Таблица А.1. Примеры синтаксиса EBNS

Пример синтаксиса	Объяснение
<Fish>	Все величины заключаются в угловые скобки
Fish	Представляет собой ключевое слово, которое должно появиться в символьном виде (регистр не имеет значения). Запятые и знаки косой черты также должны появиться в символьном виде
A B C	Сначала должно идти выражение A, затем B, затем C, именно в таком порядке
A | B	Означает альтернативу. Должно появиться либо выражение A, либо B.
A || B	Следует учесть, что это написание не соответствует настоящей EBNF форме. Должно появиться либо выражение A, либо B, либо оба сразу в любом порядке
[Fish]	Квадратные скобки используются для группировки членов
Fish*	Выражение Fish повторяется ноль или более раз. (Чистая EBNF форма)
Fish+	Выражение Fish повторяется один или больше раз. (Чистая EBNF форма)
Fish?	Выражение Fish необязательное. (Чистая EBNF форма)
Fish{A,B}	Следует учесть, что это написание не соответствует настоящей EBNF форме. Выражение Fish должно быть повторено как минимум A раз и максимум B раз

Спецификация CSS2

Спецификация CSS2 использует модифицированный синтаксис EBNS. С ней можно ознакомиться по адресу:

http://www.w3.org/TR/PR- CSS2/

Грамматика находится по адресу:

http://www.w3.org/TR/PR- CSS2/grammar.html

Если вы понимаете синтаксис EBNF, у вас не должно возникнуть трудностей при чтении спецификации или в понимании грамматики.

Приложение B
XML-ресурсы и ссылки

В сети Internet можно найти много полезных XML-ресурсов. Некоторые из них перечислены в этом приложении. В начале приведен ряд сайтов по общим вопросам, далее представлены сайты, имеющие отношения непосредственно к главам данной книги.

Сайт консорциума W3C содержит все спецификации, относящиеся к расширяемому языку разметки (XML), а также ряд других полезных ресурсов:

http://www.w3.org

Компания Microsoft помещает постоянно возрастающее число XML-ресурсов на свой сайт Site Builder. Вы можете загрузить из Сети несколько XML-анализаторов с сайта компании:

http://www.microsoft.xml

На сайте Тима Брея (Tim Bray) содержится немало ресурсов и ссылок на другую полезную информацию. Именно оттуда вы сможете загрузить анализаторы Тима, Lark и Larval:

http://www.textuality.com/XML/

На сайте Питера Флинна (Peter Flynn) вы найдете исчерпывающие ответы на часто задаваемые вопросы:

http://www.ucc.ie/xml/

Джеймс Тобер (James Tauber) поддерживает три сайта, посвященных языку XML:

http://www.xmlinfo.com – общая информация по языку XML;
http://www.schema.net – определения типа документа (DTD) и другие схемы;
http://www.xmlsoftware.com – программное обеспечение, связанное с XML.

Компания ArborText интенсивно вовлечена в разработку спецификации языка XML. На сайте компании вы найдете новости, ресурсы и ссылки:

http://www.arbortext.com/xml.html

Сайт XML.com – впечатляющая коллекция XML-ресурсов, включая очень полезную комментированную версию спецификации языка XML, созданную Тимом Бреем (Tim Bray):

http://www.xml.com/

Лайза Рейн (Lisa Rein) поддерживает обширный сайт с огромным числом ресурсов языка XML, включая ответы на часто задаваемые вопросы, ссылки на несколько анализаторов и другие ресурсы XML:

http://www.finetuning.com/

На сайте WebDeveloper.com находятся XML-файлы с учебными пособиями, ссылками и обсуждениями:

http://webdeveloper.com/xml/

Сайт Cafe Con Leche – крупный источник новостей и информации, касающейся XML:

http://sunsite.unc.edu/xml/

Сайт CommerceNet's XML Exchange – источник описаний типа документа (DTD) и других схем, предоставляемых в общее пользование:

http://www.xmlx.com/

Корпорация Poet Software поддерживает сайт, на котором находятся полезные ссылки на статьи, посвященные языку XML:

http://www.poet.com/xml/

Робин Ковер (Robin Cover) владеет списком XML-ресурсов:

http://www.oasis-open.org/cover/xml.html

Группа XML/EDI продвигает технологию EDI как XML-приложение:

http://www.xmledi.com/

Узнать все о ламах – удивительных созданиях, упомянутых во второй главе и изображенных на страницах-разделителях между главами, – можно по адресу:

http://www.llamaweb.com
http://www.frolic.org

Глава 3. XML-схемы

Сообщение по схеме XML-данных (XML-Data) помещено по адресу:

http://www.w3.org/TR/1998/NOTE-XML-data/

Сообщение по определению содержания документа (Document Content Definition) можно найти по адресу:

http://www.w3.org/TR/NOTE-dcd

Сообщение по схеме для объектно-ориентированного XML (Schema for Object-oriented XML) находится по адресу:

http://www.w3.org/TR/NOTE-SOX

Глава 4. Пространства имен

Рабочий документ по пространствам имен. Эта глава основана на версии от 16 сентября 1998 года:

http://www.w3.org/TR/WD-xml-names

Web-страница Федерального авиационного агентства содержит некоторую действительно интересную (но не имеющую отношения к XML) информацию:

http://www.faa.gov

Если вы сноб в душе или просто хотите больше узнать о британской аристократии, то эта страничка для вас бесценна!

http://www.baronage.co.uk

Рабочий проект документа по формату описания ресурсов (Resource Description Framework, RDF) расположен по адресу:

http://www.w3.org/TR/WD-rdf-syntax

Рабочий проект документа по XSL находится по адресу:

http://www.w3.org/TR/WD-xsl

Глава 5. Ссылки и указатели в XML

Принципы разработки расширяемого языка создания ссылок (Extensible Linking Language, XLL):

http://www.w3.org/TR/NOTE-xlink-principles

Спецификация X-ссылок (Xlink):

http://www.w3.org/TR/WD-xlink

Спецификация X-указателей (Xpointer):

http://www.w3.org/TR/WD-xptr

Внимание: указанные выше URL-адреса относятся к новейшим версиям спецификаций. Пятая глава данной книги основана на мартовской спецификации, которая считалась наиболее современной в момент написания книги (октябрь 1998 года).

XML-страницы Робина Ковера (Robin Cover) содержат всевозможные ссылки:

http://www.oasis-open.org/cover/

Интересная лекция с использованием 176 слайдов, подготовленная Евой Малер (Eve Mahler), соредактором спецификации XLL:

http://www.oasis-open.org/cover/xlink9805/index.htm

Как язык XML-указателей (XLink и Xpointer) может разрешить проблемы висячих ссылок в сети? Почта от Евы Мейлер:

http://www.oasis-open.org/cover/maler980331.html

Глава 6. Объектная модель XML-документа

Рекомендации консорциума W3C по объектной модели документа (DOM) можно найти по адресу:

http://www.w3.org/TR/REC-DOM-Level-1

Глава 7. Просмотр XML

Каскадные таблицы стилей

Спецификацию CSS1 (каскадные таблицы стилей) можно найти по адресу:

http://www.w3.org/TR/REC-CSS1

Спецификацию CSS2 можно найти по адресу:

http://www.w3.org/TR/REC-CSS2

Вопросы связывания XML-документов с таблицей стилей можно найти в виде сообщения по адресу:

http://www.w3.org/TR/NOTE-xml-stylesheet

Новое предложение языка стилей – Spice

Краткий обзор по этой теме. Ознакомьтесь с ним до чтения сообщения:

http://www.w3.org/People/Raggett/spice

Исходное сообщение по языку стилей Spice представлено на сайте консорциума W3C:

http://www.w3.org/TR/1998/NOTE-spice-19980123.html_

Сравнение языков XSL и Spice:

http://www.sil.org/sgml/spice-XSL980224.html

Языки DSSSL или XS

Справочная литература и учебное пособие по языку DSSSL, написанные автором этой главы, включая заметки по использованию JADE:

http://www.hypermedic.com/style

Учебное пособие Пола Прескода (Paul Prescod) по языку DSSSL. Очень рекомендуем прочитать, даже если вы не собираетесь углубляться в изучении этой темы:

http://itrc.uwaterloo.ca/~papresco/dsssl/tutorial.html

Учебное пособие Даниэля М. Джермана (Daniel M. German) предлагает несколько более детальный подход по сравнению с трудом Пола Прескода. Еще одно великолепное пособие:

http://www.sil.org/sgml/dssslGerman.html

Домашняя страница Джеймса Кларка (James Clark) с многочисленными ссылками, относящимися к DSSSL:

http://www.jclark.com/dsssl

Спецификацию языка DSSSL-Online можно найти по адресу:

http://sunsite.unc.edu/pub/sun-info/standards/dsssl/dssslo/do960816.htm

Все, какие только можно пожелать, ссылки на схемы расположены по адресу:

http://www-swiss.ai.mit.edu/scheme-home.html

Глава 9. Расширяемый язык таблиц стилей

Эти страницы содержат последние новости с фронта XSL:

http://www.w3.org/Style/XSL/

Список требований к языку XSL можно найти по адресу:

http://www.w3.org/TR/WD-XSLReq

Здесь находится рабочий проект спецификации языка XSL:

http://www.w3.org/TR/WD-xsl

Сообщение по пространству имен CSS можно найти на сайте:

http://www.w3.org/TR/NOTE-XSL-and-CSS

Глава 13. Формат определения канала

Сеть SiteBuilder компании Microsoft:

http://www.microsoft.com/sitebuilder

Пакет The Microsoft Online Internet Client SDK:

http://www.microsoft.com/msdn/sdk/inetsdk/help/default.htm

Приложение C
Спецификация расширяемого языка разметки XML 1.0

Материал приложения заимствован из рекомендации консорциума W3C от 10 февраля 1998[1]. Рекомендация доступна по адресу:

http://www.w3.org/TR/REC-xml

Редакторы:

Тим Брей (Tim Bray, Textuality и Netscape) tbray@textuality.com
Жан Паоли (Jean Paoli, Microsoft) jeanpa@microsoft.com
К.М. Сперберг-МакКуин (C.M. Sperberg-McQueen, University of Illinois at Chicago) cmsmcq@uic.edu

Аннотация

Расширяемый язык разметки (Extensible Markup Language, сокращенно называемый XML) представляет собой подмножество языка SGML, полностью описываемое в данном документе. Цель XML состоит в том, чтобы использование, передача и обработка универсального языка SGML были столь же доступны в Web, как и HTML. Язык XML был разработан для облегчения реализации и для совместимости как с SGML, так и с HTML.

Статус документа

Представленная спецификация была рассмотрена членами консорциума W3C и другими заинтересованными сторонами и одобрена Директором в качестве рекомендации консорциума W3C[2]. Это основополагающий и узаконенный документ, его можно использовать в качестве справочного материала или цитировать в качестве норматива. Роль консорциума в создании рекомендации заключалась в том, чтобы привлечь внимание к этой спецификации и обеспечить ей широкое распространение. Подобные меры направлены на повышение функциональности и дальнейшее совершенствование совместимости в Web.

---

[1] Поскольку приводимая спецификация является официальным документом, ее оформление полностью соответствует предложенному издательством Wrox. (*Прим. ред.*)

[2] Это относится только к оригинальному документу. *(Прим. ред.)*

Данный документ определяет синтаксис, предназначенный для использования в World Wide Web. Изложенный в рекомендации синтаксис является подмножеством существующего, широко используемого международного стандарта обработки текстов (Standardized Generalized Markup Language – Стандартный обобщенный язык разметки, ISO 8879:1986(E), переработанного и исправленного). Он является продуктом W3C XML Activity, подробные сведения о которой можно найти на http://www.w3.org/XML. Список текущих рекомендаций консорциума W3C и другие технические документы находятся на http://www.w3.org/TR.

В данной спецификация используется термин URI, определенный Бернерсом-Ли и другими авторами ([Berners-Lee et al.]); предполагается работа по исправлению [IETF RFC1738] и [IETF RFC1808].

Список ошибок, обнаруженных в данной спецификации, приведен на http://www.w3.org/XML/xml-19980210-errata.

О найденных ошибках сообщайте, пожалуйста, по следующему адресу: xml-editor@w3.org.

Расширяемый язык разметки XML 1.0

Содержание

1. Введение

Расширяемый язык разметки (Extensible Markup Language), сокращенно XML, описывает класс объектов данных, называемых XML-документами, а также частично описывает поведение компьютерных программ, обрабатывающих эти объекты. Язык XML – это прикладной срез *стандартного обобщенного языка*

разметки (Standardized Generalized Markup Language, SGML [ISO 8879]), или его усеченная форма. По своей конструкции XML-документы соответствуют SGML-документам.

XML-документ состоит из единиц хранения, называемых компонентами. Компоненты содержат либо разбираемые, либо неразбираемые данные. Разбираемые данные состоят из символов, некоторые из них составляют символьные данные, а некоторые образуют разметку. В разметке закодировано описание структуры хранения документа и его логическая структура. XML предоставляет механизм для наложения ограничений и на логическую конструкцию, и на структуру хранения.

Программный модуль, называемый *XML-процессором* (XML processor), используется для чтения XML-документов и предоставления доступа к их содержанию и структуре. Предполагается, что XML-процессор действует от имени другого модуля, называемого *приложением* (application). Данная спецификация описывает требуемое поведение XML-процессора с точки зрения того, как он должен считывать XML-данные и какую информацию обязан предоставлять приложению.

1.1. Цели и происхождение

Язык XML был разработан рабочей группой по XML (XML Working Group) – первоначально известной как редакционно-экспертная коллегия по SGML (SGML Editorial Review Board), – созданной под покровительством Консорциума World Wide Web (W3C) в 1996 году. Этой группой руководил Йон Босак (Jon Bosak) из фирмы Sun Microsystems. В работе активное участие принимала также специальная группа по XML (XML Special Interest Group) – ранее известная как Рабочая группа по SGML (SGML Working Group), – также организованная консорциумом W3C. Список членов рабочей группы по XML приведен в дополнении. Дэн Конноллли (Dan Connolly) осуществлял контакт между рабочей группой и консорциумом W3C.

При создании и продвижении языка XML ставились следующие цели:

1. XML должен быть языком, непосредственно используемым в Internet.
2. XML должен поддерживать самые разнообразные приложения.
3. XML должен быть совместим с SGML.
4. Написание программ, обрабатывающих XML-документы, не должно создавать дополнительные трудности.
5. Число необязательных возможностей в XML должно быть сведено к абсолютному минимуму, в идеале к нулю.
6. XML-документы должны быть удобочитаемыми и достаточно ясными.
7. Конструкция XML должна быть быстро разрабатываемой.
8. Конструкции XML должны быть краткими и формальными.
9. Создание XML-документов не должно быть трудновыполнимой задачей.
10. Краткость XML-разметки не является существенным параметром.

Эта спецификация, совместно с другими стандартами:

- Unicode и ISO/IEC 10646 для символов;
- Internet RFC 1766 для тэгов, идентифицирующих язык;

- ISO 639 для кодов названия языков;
- ISO 3166 для кодов названий стран,

предоставляет полную информацию, необходимую для понимания версии 1.0 языка XML и для построения компьютерных программ, обрабатывающих XML.

Разрешено свободное распространение данной версии спецификации XML при условии, что ее текст останется неизменным, а уведомления об авторских правах будут сохранены.

1.2. Терминология

Терминология, используемая для описания XML-документов, определяется в основной части представляемой спецификации. При подготовке основной терминологии и описании операций XML-процессора использовались термины, перечисленные ниже.

may – может.

Соответствующие документы и XML-процессоры могут вести себя как описано, но подобное поведение не является обязательным.

must – должен.

Соответствующие документы и XML-процессоры обязаны вести себя как описано; в противном случае возникает ошибка.

error – ошибка.

Нарушение правил данной спецификации; результаты не определены. Соответствующее программное обеспечение может обнаруживать ошибку и сообщать о ней, а также выходить из состояния ошибки.

fatal error – неисправимая ошибка.

Ошибка, которую соответствующий XML-процессор обязан обнаружить и уведомить о ней приложение. После обнаружения неисправимой ошибки процессор может продолжить обработку данных, чтобы найти следующие ошибки и сообщить о них приложению. С целью обеспечения возможности исправления ошибок процессор может передавать приложению необработанные данные из документа (смесь символьных данных и разметки). Однако если обнаруженная ошибка относится к категории неисправимых, процессор не должен продолжать нормальную обработку данных (то есть обязан прекратить нормальную передачу приложению символьных данных и информации о логической структуре документа).

at user option – по выбору пользователя.

Соответствующая программа может или должна (в зависимости от глагола, используемого в конкретном определении) осуществлять заданные действия. При этом должен быть предусмотрен механизм, при помощи которого пользователь мог бы запретить или разрешить выполнение этих действий.

validity constraint, VC – ограничение по состоятельности.

Правило, применяемое ко всем состоятельным XML-документам. Нарушения ограничений по состоятельности являются ошибками. Верифицирующий XML-процессор должен по выбору пользователя сообщать об этих ошибках.

well-formedness constraint, WFC – ограничение по правильности.

Правило, применяемое ко всем правильным XML-документам. Нарушения ограничений по правильности являются неисправимыми ошибками.

match – соответствовать.

Применяется:

- для строк (string) и имен (name). Две сравниваемые строки (два имени) должны быть идентичны. Символы, имеющие различное представление в стандарте ISO/IEC 10646 (например, символы, имеющие как заранее созданную форму, так и форму, образованную из основы и диакритического символа) идентичны только в том случае, если они имеют одинаковое представление в обеих строках. По выбору пользователя процессор может приводить эти символы к канонической форме. Изменение регистра не производится;
- для строк и правил в грамматике. Строка соответствует грамматическому правилу вывода, если она принадлежит языку, созданному этим правилом вывода;
- для содержания и моделей содержания. Элемент соответствует своему описанию, когда он соответствует образу, описанному в ограничении «состоятельный элемент» (Element Valid).

for compatibility – для совместимости.

Особенность языка XML, введенная исключительно с целью сохранения совместимости XML с SGML.

for interoperability – для взаимодействия.

Необязательная рекомендация, введенная для в надежде на расширение возможностей, с помощью которых XML-документы обрабатывались бы посредством существующей установленной основы SGML-процессоров, предшествовавших принятию приложения об адаптации WebSGML к стандарту ISO 8879 (WebSGML Adaptation Annex to ISO 8879).

2. Документы

Объект данных является *XML-документом* (XML document), если он правилен в том смысле, как это определено в данной спецификации. Правильный XML-документ, кроме того, может являться состоятельным, если он удовлетворяет некоторым ограничениям более высокого ранга.

У каждого XML-документа имеется как логическая, так и физическая структура. Физически документ составлен из неких единиц, называемых *компонентами* (entities). Компонент может ссылаться на другие компоненты, чтобы инициировать подстановку последних в документ. Документ начинается в *корне* (root) или, другими словами, в *компоненте документа* (document entity). Логически документ состоит из *описаний* (declaration), *элементов* (element), *комментариев* (comment), *ссылок на символы* (character reference) и *команд приложения* (processing instruction), которые указаны в документе явной разметкой. Логическая и физическая структуры должны быть правильно вложены, как описано в подразделе 4.3.2 «Правильные разбираемые компоненты».

2.1. Правильные XML-документы

Текстовый объект является *правильным* (well-formed) XML-документом, если:

1. Рассматриваемый как целое, он соответствует правилу вывода, обозначенному **document** (документ).
2. Он удовлетворяет всем ограничениям по правильности, приведенным в данной спецификации.
3. Каждый из разбираемых компонентов, на который внутри документа существует прямая или косвенная ссылка, также является правильным.

Документ (document)

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

Соответствие правилу вывода `document` подразумевает, что:

1. Документ содержит один или больше элементов.
2. Существует в точности один элемент, называемый *корнем* (root), или *элементом документа* (document element), никакая часть которого не появляется в содержании каких-либо других элементов. Для всех остальных элементов, если открывающий тэг находится в содержании другого элемента, то и закрывающий тэг находится в содержании того же элемента. Проще говоря, элементы, границы которых установлены открывающим и закрывающим тэгами, должны быть правильно вложены друг в друга.

Как следствие: для каждого некорневого элемента `C` в документе существует один элемент `P` такой, что элемент `C` находится в содержании элемента `P`, но не входит в содержание какого-либо другого элемента, включенного в содержании элемента `P`. Элемент `P` называют *родительским элементом* (parent) элемента `C`, а `C` называют *дочерним элементом* или *потомком* (child) элемента `P`.

2.2. Символы

Обрабатываемый компонент содержит *текст* (text), который может являться или разметкой, или символьными данными. *Символ* (character) представляет собой элементарную единицу текста, как определено в стандарте ISO/IEC 10646 [ISO/IEC 10646]. К допустимым символам относятся символ табуляции, символ возврата каретки, символ перевода строки и допустимые графические символы стандартов Unicode и ISO/IEC 10646. Использование *символов совместимости* (compatibility characters), как указано в разделе 6.8 [Unicode], не одобряется.

Диапазон символов (Character Range)

```
                    #x9 | #xA | #xD | [#x20-#xD7FF]   /* Любой символ Unicode
[2]  Char  ::=      | [#xE000-#xFFFD]                 за исключением блока
                    | [#x10000-#x10FFFF]             заменителей, FFFE и FFFF.*/
```

Механизм для кодировки точек кода символа в битовую комбинацию может варьироваться от компонента к компоненту. Все XML-процессоры обязаны принимать кодировки UTF-8 и UTF-16 стандарта 10646; механизмы для указания,

какая из двух кодировок используется, или для активизации других кодировок обсуждаются в разделе 4.3.3 «Кодировка символов в компонентах».

2.3. Общие синтаксические конструкции

В этом разделе определяются некоторые символы, широко используемые в грамматике.

Символ `S`, обозначающий *пробельные литеры* (white space), состоит из одного или более пробелов (`#x20`), символов возврата каретки, символов перевода строки или символов табуляции.

Пробельные литеры (White Space)

```
[3]  S      ::=      (#x20 | #x9 | #xD | #xA)+
```

Для удобства символы подразделяются на *буквы* (letter), *цифры* (digit) и прочие символы. Буквы состоят или из алфавитных символов, или из символов силлабической основы, за которыми, возможно, следуют один или более *соединяющих символов* (combining char), или из идеографических символов. Полное описание конкретных символов дано в дополнении B «Классы символов».

Имя (name) – это обозначение, начинающееся с буквы или одного из нескольких знаков пунктуации, за которым следуют буквы, цифры, дефисы, символы подчеркивания, двоеточия или точки, которые все вместе называются символами имени. Имена, начинающиеся со строки "`xml`" или любой другой строки, соответствующей выражению `(('X' | 'x') ('M' | 'm') ('L' | 'l'))`, зарезервированы для целей стандартизации в этой или последующих версиях данной спецификации.

Примечание *Символ двоеточия в именах XML зарезервирован для экспериментов с пространствами имен. Ожидается, что его значение будет стандартизировано в будущем; при этом, возможно, придется обновить документы, использующие символ двоеточия в экспериментальных целях. (Нет никаких гарантий, что какой-либо механизм пространства имен, одобренный для XML, действительно будет использовать двоеточие в качестве разделителя пространства имен.)*

На практике это означает, что разработчикам не следует использовать двоеточие в именах XML за исключением экспериментов с пространством имен, но XML-процессоры должны воспринимать двоеточие в качестве символа имени.

Идентификатор (`Nmtoken`, name token) – это любое сочетание символов имени.

Имена и обозначения (Names and Tokens)

```
[4]  NameChar ::=  Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar | Extender
[5]  Name     ::=  (Letter | '_' | ':') (NameChar)*
[6]  Names    ::=  Name (S Name)*
[7]  Nmtoken  ::=  (NameChar)+
[8]  Nmtokens ::=  Nmtoken (S Nmtoken)*
```

Символьная строка, литерал (literal data, literal) – это любая строка, заключенная в кавычки и не содержащая символов кавычек, используемых в качестве ограничителей этой строки. Символьные строки используются для определения содержания внутренних компонентов (`EntityValue`), значений атрибутов (`AttValue`) и внешних идентификаторов (`SystemLiteral`). Следует учесть, что выражение `SystemLiteral` может анализироваться без просмотра разметки.

Литералы (Literals)

```
[9]  EntityValue    ::= ' " ' ([^%&"] | PEReference | Reference)* ' " '
                      | " ' " ([^%&"] | PEReference | Reference)* " ' "
[10] AttValue       ::= ' " ' ([^%&"] | Reference)* ' " ' | " ' " ([^%&"] | Reference)* "  '  "
[11] SystemLiteral  ::= (' " ' [^"]* ' " ') | (" ' " [^']* " ' ")
[12] PubidLiteral   ::= ' " ' PubidChar* ' " ' | " ' " (PubidChar - " ' ")* " ' "
[13] PubidChar      ::= #x20 | #xD |#xA | [f-zA-Z0-9] | [-'()+,./:=?;!*#@$_%]
```

2.4. Символьные данные и разметка

Текст состоит из сочетания символьных данных и разметки. *Разметка* (markup) может принимать форму открывающих тэгов, закрывающих тэгов, тэгов пустых элементов, ссылок на компоненты, ссылок на символы, комментариев, ограничителей секций CDATA, описаний типа документа и команд приложения.

Текст, не являющийся разметкой, составляет *символьные данные* (character data) документа.

Знак амперсанда (`&`) и левая угловая скобка (`<`) могут появляться в символьной форме только в том случае, если они используются в качестве ограничителей разметки или в комментариях, командах приложения или в секциях CDATA. Их также можно применять внутри значения символьного компонента в описании внутреннего компонента (см. 4.3.2 «Правильные разбираемые компоненты»). Если эти символы необходимы в других местах, следует использовать либо численные ссылки на символы, либо строки `"&"` и `"<"` соответственно.

Правая угловая скобка (`>`) может быть представлена строкой `">"`. Для совместимости следует избегать написания правой угловой скобки в символьном виде, используя либо выражение `">"` либо ссылку на символ в том случае, когда в содержании правая угловая скобка используется в строке `"]]>"`, если данная строка не обозначает конец секции CDATA.

В содержании элементов символьными данными является любая строка символов, не содержащая открывающий ограничитель любой разметки. В секции CDATA символьными данными является любая строка символов, не включающая в себя закрывающий ограничитель секции CDATA (`"]]>"`).

Чтобы использовать как одинарные, так и двойные кавычки в значении атрибута, апостроф или символ одинарных кавычек следует представлять в виде строки `"&apos;"`, а двойные кавычки – в виде `"&quot;"`.

Символьные данные (Character Data)

```
[14] CharData   ::= [^<&]* - ([^<&]* ']]>' [^<&]*)
```

2.5. Комментарии

Комментарии (comment) можно использовать в любом месте документа вне другой разметки; кроме того, комментарии можно использовать внутри описания типа документа в местах, допустимых грамматикой. Комментарии не являются частью символьных данных документа. XML-процессор может, но не обязан, позволять приложению получать текст комментариев. Для совместимости строка, содержащая два дефиса (`"--"`), не должна использоваться внутри комментария.

Комментарии (Comments)

```
[15] Comment  ::=  <!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
```

Пример комментария:

```
<!-- declaration for <head> & <body> -->
```

2.6. Команды приложения

Команды приложения (Processing instructions, PI) позволяют документу содержать команды для приложения.

Команды приложения (Processing instructions)

```
[16] PI        ::=  '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>'
[17] PITarget  ::=  Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
```

Команды приложения не являются частью символьных данных документа, но приложению они должны быть переданы в неизменном виде. Команды приложения начинаются с указания цели (`PITarget`), используемой для обозначения приложения, которому предназначаются эти команды. Такие названия цели, как `"XML"`, `"xml"` и им подобные, зарезервированы для целей стандартизации в этой или будущих версиях представленной спецификации. Для формального описания целей команд приложения можно использовать *механизм обозначений* XML (XML Notation mechanism).

2.7. Секции CDATA

Секции CDATA (CDATA sections) можно использовать в тех же случаях, что и символьные данные. Секции CDATA применяются, чтобы избежать написания текстовых блоков, которые могут быть приняты за разметку. Секции CDATA начинаются со строки `"<![CDATA ["` и заканчиваются строкой `"]]>"`.

Разделы CDATA (CDATA sections)

```
[18] CDSect   ::=  CDStart CData CDEnd
[19] CDStart  ::=  '<![CDATA['
[20] CData    ::=  (Char* - (Char* ']]>' Char*))
[21] CDEnd    ::=  ']]>'
```

В секции CDATA только строка `CDEnd` идентифицируется как разметка, поэтому левые угловые скобки и знак амперсанда можно использовать в буквенной форме. Не следует (и такой возможности нет) заменять эти символы строками `"<"` и `"&amp"`. Секции CDATA не могут быть вложенными.

Ниже приведен пример секции CDATA, в которой выражения "<greeting>" и "</greeting>" идентифицируются как символьные данные, а не разметка.

```
<![CDATA[<greeting>Hello, world!</greeting>]]>
```

2.8. Пролог и описание типа документа

XML-документы могут и должны начинаться с *объявления XML* (XML declaration), которое определяет используемую версию языка XML. Например, приведенный ниже текст является законченным XML-документом, правильным, но не состоятельным:

```
<?xml version='1.0"?>
<greeting>Hello, world!</greeting>
```

так же как и эта строка:

```
<greeting>Hello, world!</greeting>
```

Номер версии "1.0" следует использовать для указания, что документ соответствует данной версии спецификации; указание величины "1.0" в случае несоответствия документа данной версии спецификации приведет к ошибке. В планы Рабочей группы по XML входит обозначение последующих версий данной спецификации номерами, следующими за "1.0", но эти планы не дают никаких гарантий, что последующие версии все-таки будут выпущены. Более того, нет гарантии использования некой определенной на сегодняшний день (например, предложенной выше) схемы нумерации. Поскольку появление более поздних версий не исключено, данная конструкция при необходимости предоставит возможность автоматического определения используемой версии. Процессоры могут выдавать сообщение об ошибке, если получат документы, обозначенные версией, которую они не поддерживают.

Функция разметки в XML-документе заключается в описании структуры хранения и логической структуры документа, а также в ассоциировании пар атрибут-значение с логической структурой документа. XML предоставляет механизм – *описание типа документа* (document type declaration), – который используется для определения условий соответствия логической структуры документа установленным правилам. Также при помощи описания типа документа поддерживается применение ранее определенных единиц хранения. XML-документ является *состоятельным* (valid), если имеется ассоциированное с ним описание типа документа и если документ составлен в соответствии с условиями или *ограничениями*, выраженными в этом описании.

Пролог (Prolog)

[22]	prolog	::=	XMLDecl? Misc* (doctypedecl Misc*) ?
[23]	XMLDecl	::=	<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'
[24]	VersionInfo	::=	S 'version' Eq (' VersionNum ' | " VersionNum ")
[25]	Eq	::=	S? '=' S?
[26]	VersionNum	::=	([a-zA-Z0-9_.:] | '-')+
[27]	Misc	::=	Comment | PI | S

Описание типа документа XML содержит или указывает на описания разметки, которые предоставляют грамматику для класса документов. Эта грамматика известна как *определение типа документа* (Document Type Definition, DTD). Описание типа документа может указывать на внешнее подмножество (специальный тип внешних компонентов), содержащее описание разметки; оно может также содержать описание разметки непосредственно во внутреннем подмножестве или то и другое одновременно. DTD для документа состоит из обоих подмножеств, взятых вместе.

Описание разметки (markup declaration) представляет собой описание типа элемента, описание списка атрибутов, описание компонента (EntityDecl) или описание нотации. Эти описания могут частично или полностью содержаться в параметрическом компоненте, как это описано ниже в ограничениях по правильности и по состоятельности. Более полная информация содержится в разделе 4 «Физические структуры».

Определение типа документа (Document Type Definition)

[28] doctypedecl	::= <!DOCTYPE' S Name (S ExternalID)? S? ('[' (markupdecl | PEReference | S)* ']' S?)? '>'	[VC: тип корневого элемента]
[29] markupdecl	::= Elementdecl | AttlistDecl | EntityDecl | NotationDecl | PI | Comment	[VC: состоятельное описание/вложение параметрических компонентов]
		[WFC: параметрические компоненты во внутреннем подмножестве]

Описания разметки могут быть частично или полностью составлены из *текста замены* (replacement text) параметрических компонентов. Ниже в данной спецификации под правилами вывода для нетерминальных символов (`elementdecl`, `AttlistDecl` и им подобных) понимаются описания *после* того, как были подставлены все параметрические компоненты.

Ограничение по состоятельности: тип корневого элемента.

Имя (`Name`) в описании типа документа должно соответствовать типу элемента корневого элемента.

Ограничение по состоятельности: правильное вложение описаний и параметрических компонентов.

Текст замены параметрического компонента и описание разметки должны быть правильно вложены относительно друг друга. А именно, если либо первый, либо последний символ описания разметки (обозначенного выше как `markupdecl`) находится в тексте замены ссылки на параметрический компонент, то оба символа должны располагаться в одном и том же тексте замены.

Ограничение по правильности: параметрические компоненты во внутреннем подмножестве.

Во внутреннем подмножестве DTD ссылка на параметрический компонент может встречаться только в тех местах, где находится описание разметки, но не внутри описания разметки. (Это ограничение не распространяется на ссылки, встречающиеся во внешних параметрических компонентах и на внешнее подмножество.)

Наподобие внутреннего подмножества, внешнее подмножество и любые внешние параметрические компоненты, на которые есть ссылка в DTD, должны состоять из последовательностей законченных описаний разметки, принадлежащих типам описаний, допускаемым нетерминальным символом `markupdecl`, перемежающихся пробельными литерами или ссылками на параметрические компоненты. Однако части содержания внешнего подмножества или внешнего параметрического компонента могут быть проигнорированы по условию за счет использования конструкции условной секции; во внутренних подмножествах это не разрешается.

Внешнее подмножество (External Subset)

```
[30] extSubset      ::= TextDecl? extSubsetDecl
[31] extSubsetDecl  ::= (markupdecl | conditionalSect | PEReference | S )*
```

Внешнее подмножество и внешний параметрический компонент также отличаются от внутреннего подмножества тем, что в них ссылки на параметрический компонент разрешены *внутри* описания разметки, а не только *между* описаниями разметки.

Ниже приведен XML-документ с описанием типа документа:

```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

Системный идентификатор `hello.dtd` дает URI определения типа документа (DTD) для данного документа.

Описание также может быть дано локально, как это сделано в приведенном ниже примере:

```
<?xml version="1.0"?>
<!DOCTYPE greeting [
   <!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```

В обоих примерах использованы и внутреннее, и внешнее подмножества, при этом считается, что внутреннее подмножество расположено перед внешним. Следовательно, описания компонентов и списков атрибутов во внутреннем подмножестве имеют приоритет над описаниями во внешнем подмножестве.

2.9. Объявление автономности документа

Описания разметки могут влиять на содержание документа, передаваемого приложению из XML-процессора; так, например, происходит со значениями атрибутов по умолчанию и с описаниями компонентов. *Объявление автономности документа* (standalone document declaration), встречающееся как компонент XML-описания, показывает, имеются ли описания, являющиеся внешними к компоненту документа.

Объявление автономности документа (Standalone Document Declaration)

```
[32] SDDecl  ::= S 'standalone' Eq (('' ' '' ('yes'        [VC: объявление автономности документа ]
                 | 'no') '' ' '') | (' '' ' ('yes' | 'no') ' '' '))
```

В объявлении автономности документа величина "yes" указывает, что не существует описаний разметки, являющихся внешними по отношению к компоненту документа (или во внешнем подмножестве DTD или во внешнем параметрическом компоненте, на который есть ссылка из внутреннего подмножества) и влияющих на информацию, передаваемую приложению из XML-процессора. Значение "no" определяет, что подобное внешнее описание разметки существует или может существовать. Следует учесть, что объявление автономности документа указывает на наличие внешних *описаний*; наличие в документе ссылок на внешние *компоненты*, если эти компоненты описаны внутренне, не меняет статуса автономности.

Если в документе нет внешних описаний разметки, объявление автономности документа не имеет смысла. Если имеются внешние компоненты разметки, а объявление автономности документа не задано, то подразумевается значение "no".

Любой XML-документ, для которого верно выражение standalone="no", допускается алгоритмически преобразовать в автономный документ, что может оказаться желательным в некоторых сетевых приложениях.

Ограничение по состоятельности: объявление автономности документа.

Значение объявления автономности документа должно быть равным "no", если любое из внешних описаний разметки содержит описания:

- атрибутов со значениями по умолчанию, если в документе элементы, к которым относятся эти атрибуты, даны без указания значений для этих атрибутов;
- компонентов (отличных от amp, lt, gt, apos, quot), если в документе имеются ссылки на эти компоненты;
- атрибутов со значениями, подлежащими нормализации, если в документе встречается атрибут со значением, которое изменится в результате нормализации;
- типов элементов с содержимым элементов, если непосредственно внутри любой реализации этих типов встречаются пробельные литеры.

Ниже приведен пример объявления XML с объявлением автономности документа:

```
<?xml version="1.0" standalone='yes'?>
```

2.10. Обработка пробельных литер

При редактировании XML-документов часто удобнее использовать *пробельные литеры* (white spaces) (пробелы, знаки табуляции, пустые строки; в данной спецификации они обозначены нетерминальным символом S), чтобы разделить разметку и этим улучшить читаемость документа. Зачастую подразумевается, что эти пробельные литеры не будут включены в окончательную версию документа. С другой стороны, бывают и «значимые» пробельные литеры, которые необходимо оставить в окончательной версии, например, в стихах или в исходных кодах.

XML-процессор обязан всегда передавать приложению все символы докумен-

та, не являющиеся разметкой. *Верифицирующий* (validating), то есть проверяющий на состоятельность, XML-процессор также обязан сообщать приложению, какие из этих символов образуют пробельные литеры, расположенные в содержании элементов.

К элементу можно присоединить специальный атрибут, называемый `xml:space`, чтобы показать намерение сохранить пробельные литеры в этом элементе при помощи приложения. В состоятельных документах при использовании данного атрибута его (как и все другие атрибуты) обязательно следует описать. При описании этот атрибут должен быть отнесен к перечисляемому типу, единственно возможными значениями которого являются `"default"` (по умолчанию) и `"preserve"` (сохранять). Например:

```
<!ATTLIST poem xml:space (default | preserve) 'preserve'>
```

Значение `"default"` сообщает, что для этого элемента пригодны методы обработки пробельных литер, используемые по умолчанию в данном приложении; значение `"preserve"` показывает, что приложению следует сохранить пробельные литеры в данном элементе. Намерение сохранять или не сохранять пробельные литеры распространяется на все элементы, находящиеся в содержании того элемента, в котором оно было декларировано, до тех пор, пока это намерение не будет переопределено другим атрибутом `xml:space`.

Считается, что корневой элемент любого документа не устанавливает, как приложению следует поступать с пробельными литерами, если не дано значение для этого атрибута или если атрибут описан со значением по умолчанию.

2.11. Обработка окончаний строк

Разбираемые компоненты XML часто хранятся в файлах, которые для удобства редактирования разбиты на строки. Обычно эти строки отделены друг от друга некоторой комбинацией символов возврата каретки (`#xD`) и перевода строки (`#xA`).

Чтобы облегчить работу приложению, во всех местах, где внешний разбираемый компонент или символьное значение компонента внутреннего разбираемого компонента содержит или последовательность двух символов `"#xD#xA"`, или отдельно стоящую символьную константу `#xD`, XML-процессор должен передать приложению только один символ `#xA`. (Этого можно добиться, нормализовав все разрывы строк в символ `#xA` на входе, перед проведением разбора.)

2.12. Обозначение языка

При обработке документов зачастую бывает полезно обозначать естественный или формальный язык, на котором написано содержание. Для обозначения языка, использованного в содержании и в значениях атрибутов любых элементов, в XML-документе можно применять специальный атрибут `xml:lang`. При употреблении этого атрибута в состоятельном документе, он, как и все другие атрибуты, должен быть описан. Величинами атрибута являются указатели языка, определенные стандартом [IETF RFC 1766] в разделе «Тэги для обозначения языка» (Tags for the Identification of Languages).

Обозначение языка (Language Identification)

```
[33] LanguageId  ::= Langcode ('-' Subcode)*
[34] Langcode    ::= ISO639Code | IanaCode | UserCode
[35] ISO639Code  ::= ([a-z] | [A-Z]) ([a-z] | [A-Z])
[36] IanaCode    ::= ('i' | 'I') '-' ([a-z] | [A-Z])+
[37] UserCode    ::= ('x' | 'X') '-' ([a-z] | [A-Z])+
[38] Subcode     ::= ([a-z] | [A-Z])+
```

Под кодом языка (`Langcode`) может подразумеваться следующее:

- двухбуквенный код языка, определенный стандартом [ISO 639] (Codes for the representation of names of language – коды для представления названий языков);
- указатель языка, зарегистрированный Комитетом по назначению параметров Internet (Internet Assigned Numbers Authority, IANA). Эти указатели начинаются с префикса `"i-"` (или `"I-"`);
- указатели языка, назначенные пользователем или согласованные между взаимодействующими сторонами для частного применения. Эти указатели должны начинаться с префикса `"x-"` или `"X-"`, чтобы гарантировать отсутствие конфликтов с названиями, которые впоследствии будут зарегистрированы или стандартизованы IANA.

Допустимо любое число сегментов *подкода* (`Subcode`). Если существует первый сегмент подкода и подкод состоит из двух букв, им должен быть код страны из стандарта [ISO 31666] (Codes for the representation of names of countries – коды для представления названий стран). Если первый подкод состоит более чем из двух букв, им должен быть подкод для языка, зарегистрированного IANA, если только код языка `Langcode` не начинается с префикса `"x-"` или `"X-"`.

Хорошим тоном считается указывать код языка в нижнем регистре, а код страны (если необходимо), в верхнем. Следует обратить внимание, что для этих величин, в отличие от других имен в XML-документах, строчные и заглавные буквы не различаются.

Например:

```
<p xml:lang="en">The quick brown fox jumps over the lazy dog.</p>
<p xml:lang="en-GB">What colour is it?</p>
<p xml:lang="en-US">What color is it?</p>
<sp who="Faust" desc='leise' xml:lang="de">
<l>Habe nun, ach! Philosophie,</l>
<l>Juristerei, und Medizin</l>
<l>und leider auch Teologie</l>
<l>durchaus studiert mit heißen Bemüh'n.</l>
</sp>
```

Считается, что значение языка, выставляемое атрибутом `xml:lang`, распространяется на все атрибуты и все содержание элемента, в котором определено это значение атрибута, если значение языка не переопределено в другом элементе, находящимся внутри рассматриваемого содержания.

Простое определение для атрибута `xml:lang` может принимать следующую форму:

```
xml:lang NMTOKEN #IMPLIED
```

Кроме того, если это считается уместным, можно задавать конкретные значения по умолчанию. В стихах на французском языке для английских студентов со сносками и заметками на полях, сделанными на английском языке, объявить атрибут `xml:lang` можно следующим способом:

```
<!ATTLIST poem xml:lang NMTOKEN 'fr'>
<!ATTLIST gloss xml:lang NMTOKEN 'en'>
<!ATTLIST note xml:lang NMTOKEN 'en'>
```

3. Логические структуры

Каждый XML-документ содержит один или более элементов, границы которых либо ограничены открывающим и закрывающим тэгами, либо, для пустых элементов, тэгом пустого элемента. У каждого элемента имеется тип, идентифицируемый именем, которое иногда называют *групповым идентификатором* (Generic Identifier, GI), а также может иметься набор спецификаций атрибутов. У каждой спецификации атрибута имеется имя и значение.

Элемент (Element)

[39] element ::= EmptyElemTag | STag content Etag [WFC: соответствие типа элемента] [VC: состоятельный элемент]

Эта спецификация не накладывает ограничений на семантику, на использование, а также (кроме синтаксиса) на имена типов элементов и атрибутов, за исключением тех случаев, когда имена начинаются со строки, соответствующей выражению `(('X' | 'x') ('M' | 'm') ('L' | 'l'))`, которое зарезервировано для целей стандартизации в этой или последующих версиях данной спецификации.

Ограничение по правильности: соответствие типа элемента.

Имя (`Name`) в закрывающем тэге элемента должно соответствовать типу элемента в открывающем тэге.

Ограничение по состоятельности: состоятельный элемент.

Элемент считается состоятельным, если имеется описание, соответствующее выражению `elementdecl` (описание элемента), в котором значение имени (`Name`) соответствует типу элемента, и если выполняется одно из следующих четырех условий:

1. Описание соответствует выражению `EMPTY` (пустой), и у элемента нет содержания.
2. Описание соответствует выражению `children` (дочерние), и последовательность дочерних элементов принадлежит языку, созданному регулярным выражением в модели содержания с необязательными пробельными литерами

(символами, соответствующими нетерминальному символу S) между каждой парой дочерних элементов.

3. Описание соответствует выражению Mixed (смешанный), и содержание состоит из символьных данных и дочерних элементов, типы которых соответствуют именам в модели содержания.
4. Описание соответствует выражению ANY (любой), и типы любых дочерних элементов были описаны прежде.

3.1. Тэги открывающие, закрывающие и пустого элемента

Начало каждого непустого элемента XML обозначается *открывающим тэгом* (start-tag).

Открывающий тэг (Start-tag)

[40] STag	::= <" Name (S Attribute)* S?">'	[WFC: уникальная спецификация атрибута]
[41] Attribute	::= Name Eq AttValue	[VC: тип значения атрибута] [WFC: отсутствие ссылок на внешние компоненты] [WFC: отсутствие символа "<" в значениях атрибута]

Выражение Name (имя) в открывающем и закрывающем тэгах дает *тип* (type) элемента. Пары Name-AttValue (имя-значение атрибута) называются *спецификациями атрибутов* (attribute specifications) элемента, выражение Name в каждой паре называется *именем атрибута* (attribute name), содержание AttValue (текст между ограничителями в виде одинарных или двойных кавычек) называется *значением атрибута* (attribute value).

Ограничение по правильности: уникальная спецификация атрибута.

Никакое имя атрибута не может появляться более одного раза в одном и том же открывающем тэге или тэге пустого элемента.

Ограничение по состоятельности: тип значения атрибута.

Атрибут должен быть описан; значение должно относиться к типу, объявленному для атрибута. (О типах атрибутов см. раздел 3.3 «Описания списка атрибутов».)

Ограничение по правильности: отсутствие ссылок на внешние компоненты.

Значение атрибута не может содержать прямые или косвенные ссылки на внешние компоненты.

Ограничение по правильности: отсутствие символа "<" в значениях атрибута.

Текст замены (отличный от "<") любого компонента, на который в значении атрибута есть прямая или косвенная ссылка, не должен содержать символы левой угловой скобки "<".

Пример открывающего тэга:

```
<termdef id="dt-dog" term="dog">
```

Окончание каждого элемента, начинающегося с открывающего тэга, должно быть обозначено *закрывающим тэгом* (end-tag), содержащим имя (name), которое повторяет тип элемента, указанный в открывающем тэге.

Закрывающий тэг (End-tag)

[42] ETag ::= </' Name S? '>'

Пример закрывающего тэга:

```
</termdef>
```

Текст, находящийся между открывающим и закрывающим тэгами, называется *содержанием* (content) элемента:

Содержание элементов (Content of Elements)

[43] Content ::= (element | CharData | Reference | CDSect | PI | Comment)*

Если элемент *пустой* (empty), то он должен быть обозначен либо открывающим тэгом, за которым немедленно следует закрывающий тэг, либо тэгом пустого элемента. *Тэг пустого элемента* (empty-element tag) имеет специальную форму:

Тэги для пустых элементов (Tags for Empty Elements)

[44] EmptyElemTag ::= <' Name (S Attribute)* S? '/>' [WFC: уникальная спецификация атрибута]

Тэги пустого элемента можно использовать для любого элемента, не имеющего содержания, вне зависимости от того, был ли этот элемент описан с использованием ключевого слова `EMPTY`. Для обеспечения взаимодействия тэги пустого элемента должны применяться только для элементов, которые описаны с применением ключевого слова `EMPTY`.

Примеры пустых элементов:

```
<IMG align="left"
src="http://www.w3.org/Icons/WWW/w3c_home" />
<br></br>
<br/>
```

3.2. Описания типа элемента

Для того чтобы соответствовать критерию состоятельности, структура элементов в XML-документе может быть ограничена за счет использования описаний типа элементов и описаний списка атрибутов. Описание типа элементов накладывает ограничения на содержание элементов.

Часто описания типа элементов ограничивают, какие типы элементов могут использоваться в элементе в качестве дочерних элементов. По выбору пользователя XML-процессор может выдавать предупреждение, когда в описании упоминается тип элемента, для которого не имеется описания, однако подобная ситуация не является ошибкой.

Описание типа элемента (element type declaration) имеет форму, где значение имени (`Name`) дает тип описываемого элемента.

Описание типа элемента (Element Type Declaration)

[45] elementdecl ::= '<!ELEMENT' S Name S [VC: уникальное описание типа элемента]
contentspec S? '>'

[46] contentspec ::= 'EMPTY' | 'ANY' | Mixed | children

Ограничение по состоятельности: уникальное описание типа элемента. Ни один тип элементов не может быть описан более одного раза.

Примеры описаний типа элементов:

```
<!ELEMENT br EMPTY>
<!ELEMENT p (#PCDATA | emph)*>
<!ELEMENT %name.para; %content.para;>
<!ELEMENT container ANY>
```

3.2.1. Элементное содержание

В том случае, когда элементы данного типа содержат только дочерние элементы (но не символьные данные), возможно, отделенные друг от друга пробельными литерами (символами, соответствующими нетерминальному символу `S`), – говорят, что тип элементов имеет *элементное содержание* (element content). Данное ограничение включает в себя модель содержания, простую грамматику, регулирующую разрешенные типы дочерних элементов, и порядок, в котором допустимо их появление. Грамматика построена на *частицах содержания* (content particle, `cp`), состоящих из имен (`Name`), списков выбора частиц содержания (`choice`) или списков последовательностей частиц содержания (`seq`), где каждое значение `Name` представляет собой тип элементов, который может появиться в качестве дочернего.

Модель элементного содержания (Element-content model)

[47]	children	::= (choice | seq) ('?' | '*' | '+')?	
[48]	cp	::= (Name | choice | seq) ('?' | '*' | '+')?	
[49]	choice	::= '(' S? cp (S? '|' S? cp)* S? ')'	[VC: правильное вложение групп и параметрических компонентов]
[50]	seq	::= (' S? cp (S? ',' S? cp)* S? ')'	[VC: правильное вложение групп и параметрических компонентов]

Любая частица содержания в списке выбора может появиться в элементном содержании в том месте, где в грамматике появляется список выбора; каждая из частиц содержания, встречающихся в списке последовательности, должна появиться в элементном содержании согласно порядку, задаваемому этим списком. Необязательный символ, следующий за именем или списком, определяет, появляются ли элемент или частицы содержания из списка: один или больше раз (`+`), ноль или больше раз (`*`), ноль или один раз (`?`).

Отсутствие подобного символа означает, что элемент или частица содержания должны появиться точно один раз. Использованные синтаксис и значение такие же, как и в представлениях в данной спецификации.

Содержание элемента соответствует модели содержания в том и только в том случае, если есть возможность проследить путь через модель содержания, соблюдая операторы последовательности, выбора и повторения и находя соответствие между каждым элементом в содержании и типом элементов в модели содержания. Для совместимости, понимаемой именно в таком смысле, считается ошибочной ситуация, когда элемент в документе может соответствовать более чем одному вхождению типа элемента в модели содержания. Для более подробной информации обращайтесь к дополнению E «Детерминированные модели содержания».

Ограничение по состоятельности: правильное вложение групп и параметрических компонентов.

Текст замены параметрического компонента и заключенные в круглые скобки группы должны быть правильно расположены или вложены друг относительно друга. Это означает, что если либо открывающая, либо закрывающая круглая скобка в конструкциях `choice` (выбор), `seq` (последовательность) или `Mixed` (смешанный) содержится во вставляемом тексте для параметрического компонента, то обе скобки обязаны находится в одном и том же вставляемом тексте. Для взаимодействия, если ссылка на параметрический компонент появляется в конструкциях `choice`, `seq` или `Mixed`, то ее текст замены не должен быть пуст, и ни первый, ни последний непустой символ текста замены не должен быть знаком соединителя (`|` или `,`).

Примеры моделей элементного содержания:

```
<!ELEMENT spec (front, body, back?)>
<!ELEMENT divl (head, p | list | note)*, div2*)>
<!ELEMENT dictionary-body (%div.mix; | %dict.mix;)*>
```

3.2.2. Смешанное содержание

Тип элементов имеет *смешанное содержание* (mixed content), если элементы данного типа могут содержать символьные данные, возможно, перемешанные с дочерними элементами. В этом типы дочерних элементов могут быть ограничены, но на порядок или на число появлений дочерних элементов ограничений не накладывается:

Описание смешанного содержания (Mixed-content Declaration)

[51] Mixed ::=	(' S? '#PCDATA' (S? '|' S? Name)* S? ')*' | '(' S? '#PCDATA' S? ')'	[VC: правильное вложение групп и параметрических компонентов] [VC: запрет на дублирование типов]

Здесь имена (`Name`) обозначают типы элементов, которые могут быть использованы в качестве дочерних.

Ограничение по состоятельности: запрет на дублирование типов

Одно и тоже имя не может использоваться более одного раза в одном и том же описании смешанного содержания.

Примеры описаний смешанного содержания:

```
<!ELEMENT p (#PCDATA |a|ul|b|i|em)*)>
<!ELEMENT p (#PCDATA | %font; | %phrase; | %special; | %form;)*>
<!ELEMENT b (#PCDATA)>
```

3.3. Описания списка атрибутов

Атрибуты используются для сопоставления пар имя-значение с элементами. Спецификации атрибута могут появляться только внутри открывающих или закрывающих тэгов и тэгов пустого элемента. Правило вывода, использованное для их распознавания, находится в разделе 3.1 «Тэги открывающие, закрывающие и пустого элемента». Описания списка атрибутов могут применяться для:

- определения набора атрибутов, принадлежащих данному типу элемента;
- установления ограничений типов для этих атрибутов;
- предоставления для атрибутов значений по умолчанию.

Описания списка атрибутов (attribute-list declarations) указывают имя, тип данных и значение по умолчанию (если таковое имеется) каждого атрибута, сопоставленного с определенным типом элемента:

Описание списка атрибутов (Attribute-list Declaration)

```
[52] AttlistDecl  ::=  <!ATTLIST' S Name AttDef* S? '>'
[53] AttDef       ::=  S Name S AttType S DefaultDecl
```

Имя (`Name`) в правиле `AttlistDecl` представляет собой тип элемента. По выбору пользователя XML-процессор может выдавать предупреждение, если описаны атрибуты для типа элементов, который сам не был объявлен, однако это не является ошибкой. Имя `Name` в правиле `AttDef` представляет собой имя атрибута.

Когда для заданного типа элемента применено более одного описания списка атрибутов, `AttlistDecl`, содержания всех представленных описаний соединяются. Когда к одному и тому же атрибуту заданного элемента применено более одного определения, используется первое, а остальные игнорируются. Для взаимодействия разработчики определений типа документа (DTD) могут решить, что будут предоставлять не более одного описания списка атрибутов для заданного типа элемента, не более одного определения атрибута для заданного имени атрибута и не менее одного определения атрибута в каждом описании списка атрибутов. Для взаимодействия XML-процессор может, по выбору пользователя, выдавать предупреждение, когда для заданного типа элемента дано более одного описания списка атрибутов или когда для заданного атрибута предоставлено более одного определения атрибута, однако это не является ошибкой.

3.3.1. Типы атрибутов

Существует три вида типов XML-атрибутов: строковый тип (`StringType`), набор типов с ярлыками (`TokenizedType`) и перечислимый тип (`EnumeratedType`). Значением атрибута строкового типа может быть любая символьная строка; типы с ярлыками имеют различные лексические и семантические ограничения.

Типы атрибутов (Attribute Types)

```
[54] AttType          ::= StringType | TokenizedType | EnumeratedType
[55] StringType       ::= 'CDATA'
[56] TokenizedType    ::= 'ID'                [VC: идентификационный номер ]
                                              [VC: один идентификационный номер для типа элемен-
та]
                                              [VC: ID Attribute Default ]
                        | 'IDREF'             [VC: IDREF ]
                        | 'IDREFS'            [VC: IDREF ]
                        | 'ENTITY'            [VC: имя компонента ]
                        | 'ENTITIES'          [VC: имя компонента ]
                        | 'NMTOKEN'           [VC: идентификатор имени ]
                        | 'NMTOKENS'          [VC: идентификатор имени ]
```

Ограничение по состоятельности: идентификационный номер

Значения типа `ID` (идентификационный номер) должны соответствовать правилу вывода для имен (`Name`). Имя не должно появляться более одного раза в XML-документе в качестве значения этого типа, то есть значения идентификационного номера (ID) должны однозначно определять элементы, которые ими обладают.

Ограничение по состоятельности: единственный идентификационный номер для типа элемента.

Ни для одного типа элемента не может быть указано более одного атрибута идентификационного номера (атрибут ID).

Ограничение по состоятельности: значение по умолчанию атрибута идентификационного номера.

Атрибут идентификационного номера (атрибут ID) должен иметь описанное значение по умолчанию или `#IMPLIED` или `#REQUIRED`.

Ограничение по состоятельности: IDREF.

Значения типа `IDREF` должны соответствовать правилу вывода для имени (`Name`), значения типа `IDREFS` должны соответствовать правилу вывода для имен (`Names`); каждое имя должно соответствовать значению атрибута ID для некоторого элемента в XML-документе; то есть значения `IDREF` должны соответствовать значению некоторого атрибута ID.

Ограничение по состоятельности: имя компонента.

Значения типа `ENTITY` должны соответствовать правилу вывода для имени (`Name`), значения типа `ENTITIES` должны соответствовать правилу вывода для имен (`Names`); каждое имя (`Name`) должно соответствовать имени неразбираемого компонента, описанного в DTD.

Ограничение по состоятельности: идентификатор имени.

Значения типа `NMTOKEN` должны соответствовать правилу вывода для идентификатора (`Nmtoken`); значения типа `NMTOKENS` должны соответствовать правилу вывода для идентификаторов (`Nmtokens`).

Перечислимые атрибуты (enumerated attributes) могут принимать одно значение из списка значений, предоставленных в описании. Существуют два вида перечислимых типов:

Типы перечислимых атрибутов (Enumerated Attributes Type)

[57] EnumeratedType	::= NotationType | Enumeration	
[58] NotationType	::= 'Notation' S '(' S? Name (S? '|' S? Name)* S? ')'	[VC: атрибуты нотации]
[59] Enumeration	::= '(' S? Nmtoken (S? '|' S? Nmtoken)* S? ')'	[VC: перечисление]

Атрибут обозначения (`NOTATION`) определяет обозначение, описанное в DTD с ассоциированной системой и/или общедоступными (public) идентификаторами, предназначенное для интерпретации элемента, к которому присоединены атрибуты.

Ограничение по состоятельности: атрибуты обозначения.

Значения данного типа должны соответствовать одному из имен обозначения, включенных в описание. Все имена обозначений в описании должны быть определены.

Ограничение по состоятельности: перечисление.

Значения данного типа должны соответствовать одному из идентификаторов `Nmtoken` в описании. Для взаимодействия один и тот же идентификатор `Nmtoken` не должен появляться более одного раза в типах перечислимых атрибутов одного типа элемента.

3.3.2. Значения атрибутов по умолчанию

Описание элементов предоставляет информацию о том, обязательно ли наличие атрибута, и если необязательно, то указывает, как должен реагировать XML-процессор, если описанного атрибута в документе нет.

Значения атрибутов по умолчанию (Attribute Defaults)

```
[60] DefaultDecl ::= '#REQUIRED' | '#IMPLIED' [ VC: требуемый атрибут ]
                   | (('#FIXED' S)? AttValue   [ VC: допустимое значение атрибута по умолчанию]
                                               [WFC: отсутствие символа "<" в значениях атрибута]
                                               [VC: обязательное значение атрибута по умолчанию]
```

В описании атрибута ключевое слово `#REQUIRED` (обязательный) означает, что атрибут всегда должен быть предоставлен, ключевое слово `#IMPLIED` (подразумеваемый) определяет, что значение по умолчанию не предоставляется. Если в описании не указано ни `#REQUIRED`, ни `#IMPLIED`, тогда значение `AttValue` имеет описанное *значение по умолчанию* (default). Ключевое слово `#FIXED` (фиксированный) устанавливает, что атрибут всегда должен иметь значение по умолчанию. Если указано значение по умолчанию, то когда XML-процессор встречает опущенный атрибут, он ведет себя так, как будто присутствует атрибут с описанным значением по умолчанию.

Ограничение по состоятельности: обязательный атрибут.

Если описанием по умолчанию является ключевое слово `#REQUIRED`, то атрибут должен быть указан для всех элементов, относящихся к данному типу из описания списка атрибутов.

Ограничение по состоятельности: допустимое значение атрибута по умолчанию.

Описанное значение по умолчанию должно удовлетворять лексическим ограничениям типа описанного атрибута.

Ограничение по состоятельности: обязательное значение атрибута по умолчанию.

Если атрибут имеет описанное значение по умолчанию с использованием ключевого слова `#FIXED`, то экземпляры этого атрибута должны соответствовать значению по умолчанию.

Примеры описаний списка атрибутов:

```
<!ATTLIST  termdef
           id          ID              #REQUIRED
           name        CDATA           #IMPLIED>
<!ATTLIST  list
           type        (bullets|ordered|glossary)      "ordered">
<!ATTLIST  form
           method      CDATA           #FIXED "POST">
```

3.3.3. Нормализация значений атрибутов

Перед тем как значение атрибута передается приложению или проверяется на состоятельность, XML-процессор должен нормализовать его следующим образом:

- ссылка на символ обрабатывается присоединением к значению атрибута символа, на который указывает ссылка;
- ссылка на компонент обрабатывается путем рекурсивной обработки текста замены компонента;
- символ пробельной литеры (`#x20`, `#xD`, `#xA`, `#x9`) обрабатывается путем присоединения символа `#x20` к нормализованному значению, за исключением того, что вместо последовательности `#xD#xA`, являющейся частью внешнего разбираемого компонента или символьным значением компонента для внутреннего разбираемого компонента, к нормализованному значению присоединяется только один символ `#x20`;
- другие символы обрабатываются путем их присоединения к нормализованному значению.

Если описанное значение не является секцией `CDATA`, то далее XML-процессор должен обрабатывать нормализованное значение атрибута, отбрасывая любые ведущие и заключительные символы пробела (`#x20`) и заменяя последовательности символов пробела (`#x20`) на одиночный символ пробела (`#x20`).

Все атрибуты, для которых не было считано описание, должны обрабатываться неверифицирующим анализатором, как если бы они были описаны, как `CDATA`.

3.4. Условные секции

Условные секции (conditional sections) представляют собой части внешнего подмножества описания типа документа, которые или вносятся или удаляются из логической структуры определения типа документа (DTD) на основании ключевых слов, управляющих этими секциями.

Условная секция (Conditional Section)

[61] conditionalSect	::=	includeSect | ignoreSect
[62] includeSect	::=	'<![' S? 'INCLUDE' S? '[' extSubsetDecl ']]>'
[63] ignoreSect	::=	'<![' S? 'IGNORE' S? '[' ignoreSectContents* ']]>'
[64] ignoreSectContents	::=	Ignore ('<![' ignoreSectContents ']]>' Ignore)*
[65] Ignore	::=	Char* - (Char* ('<![' | ']]>') Char*)

Подобно внешнему и внутреннему подмножеству DTD условная секция может содержать одно или больше описаний, комментариев, команд программе-приложению или вложенных условных секций, перемежающихся пробельными литерами.

Если ключевым словом условной секции является `INCLUDE` (включать), то содержание условной секции представляет собой часть DTD. Если ключевым словом условной секции определяется как `IGNORE` (игнорировать), то содержание условной секции логически не является частью DTD. Следует обратить внимание, что для надежного разбора должно быть прочитано содержание даже игнорируемых сек-

ций, чтобы обнаружить вложенные условные секции и убедиться, что завершение самой внешней (игнорируемой) условной секции определено правильно. Если условная секция с ключевым словом `INCLUDE` встречается внутри большей условной секции с ключевым словом `IGNORE`, то игнорируются обе секции, и внешняя и внутренняя.

Если ключевым словом условной секции является ссылка на параметрический компонент, то параметрический компонент должен быть заменен его содержанием до того как процессор решит, следует ли включать или игнорировать условную секцию.

Пример:

```
<!ENTITY %draft 'INCLUDE' >
<!ENTITY %final 'IGNORE' >

<![%draft;[
<!ELEMENT book (comments*, title, body, supplements?)>
]]>
<![%final;[
<!ELEMENT book (title, body, supplements?)>
]]>
```

4. Физические структуры

XML-документ может состоять из одной или нескольких хранимых единиц. Эти единицы называются *компонентами* (entities), все они имеют содержание и все (кроме компонента документа, о котором смотри ниже, и внешнего подмножества DTD) идентифицируются *именем* (name). Каждый XML-документ имеет один компонент, называемый компонентом документа, который служит стартовой точкой для XML-процессора и может содержать весь документ.

Компоненты могут быть либо разбираемыми, либо неразбираемыми. Содержание *разбираемого компонента* (parsed entity) называется его текстом замены; этот текст рассматривается как неотъемлемая часть документа.

Неразбираемый компонент (unparsed entity) представляет собой ресурс, содержание которого может быть, а может и не быть текстом, а если содержание является текстом, то оно может и не быть XML-текстом. Каждый неразбираемый компонент имеет связанное с ним обозначение, идентифицируемое именем. Кроме требования, чтобы XML-процессор делал идентификаторы для компонентов и обозначений доступными для приложения, в языке XML не налагается других ограничений на содержание неразбираемых компонентов.

Разбираемые компоненты вызываются по имени с использованием ссылок на компоненты; неразбираемые компоненты вызываются по имени, данному в значениях атрибутов `ENTITY` и `ENTITIES`.

Обычные компоненты (general entities) представляют собой компоненты, предназначенные для использования внутри содержания документа. В данной спецификации обычные компоненты иногда называются просто *компонентами*, если это не приводит к неоднозначности. Параметрические компоненты параметров являются разбираемыми компонентами, предназначенными для использования в

определении типа документа (DTD). Эти два типа компонентов используют разные формы ссылок, и распознаются в разных контекстах. Более того, они занимают разные пространства имен; параметрический компонент и обычный компонент с одинаковыми именами являются различными компонентами.

4.1. Ссылки на символы и компоненты

Ссылка на символ (character reference) указывает на определенный символ в наборе ISO/IEC 10646, например, на символ, к которому нет непосредственного доступа при помощи устройств ввода.

Ссылка на символ (Character Reference)

```
[66] CharRef    ::=  '&#' [0-9]+ ';'| '&#x' [0-9a-fA-F]+ ';'   [WFC: допустимый символ ]
```

Ограничение по правильности: допустимый символ.

Символы, которые указаны при помощи ссылок, должны соответствовать правилу вывода для символа (char).

Если ссылка на символ начинается с `"&#x"`, цифры и буквы, расположенные до завершающего символа `";"`, являются шестнадцатеричным представлением кода символа в стандарте ISO/IEC 10646. Если же ссылка начинается просто с `"&#"`, цифры, расположенные до завершающего символа `";"`, дают десятичное представление кода символа.

Ссылка на компонент (entity reference) указывает на содержание названного компонента. Ссылки на разбираемые общие компоненты в качестве ограничителей используют амперсанд (`&`) и точку с запятой (`;`). *В ссылках на параметрический компонент* (paramenter-entity reference) ограничителями служат знак процента (`%`) и точка с запятой (`;`).

Ссылка на компонент (Entity Reference)

```
[67] Reference  ::=  EntityRef | CharRef
[68] EntityRef  ::=  '&' Name ';'        [WFC: описанный компонент ]
                                         [VC: описанный компонент ]
                                         [WFC: разбираемый компонент ]
                                         [WFC: запрет на рекурсию]
[68] EntityRef  ::=  %' Name ';'         [VC: описанный компонент ]
                                         [WFC: запрет на рекурсию]
                                         [WFC: в определении типа документа (DTD)]
```

Ограничение по правильности: описанный компонент.

В документе без определения типа документа (DTD), в документе только с внутренним подмножеством DTD, не содержащим ссылки на компоненты, или в документе с записью `standalone='yes'` значение имени (`Name`), даваемое в ссылке на компонент, должно соответствовать имени, указанному в описании компонента, за исключением того, что для правильных документов не требуется описывать компоненты `amp`, `lt`, `gt`, `apos`, `quot`. Описание параметрического компонента должно предшествовать любой ссылке на этот компонент. Аналогично описание обычного компонента должно предшествовать любой ссылке на него, которая появляется в значении по умолчанию в описании списка атрибутов. Следует участь, что если компоненты описаны во внешнем подмножестве или во внешних параметрических компонентах, то неверифицирующий анализатор не обязан считывать и обраба-

тывать описания этих компонентов; для подобных документов правило, заключающееся в том, что компоненты должны быть описаны, является ограничением по правильности, только если выполняется условие `standalone='yes'`.

Ограничение по состоятельности: описанный компонент.

В документе с внешним подмножеством или с внешними параметрическими компонентами при `standalone='no'` значение имени (`Name`), даваемое в ссылке на компонент, должно соответствовать имени, указанному в описании компонента. Для взаимодействия состоятельный документ должен описывать компоненты `amp`, `lt`, `gt`, `apos`, `quot` в форме, указанной в разделе 4.6 «Предопределенные компоненты». Описание параметрического компонента должно предшествовать ссылке на него. Аналогично описание обычного компонента должно предшествовать любой ссылке на него, которая появляется в значении по умолчанию в описании списка атрибутов.

Ограничение по правильности: разбираемый компонент.

Ссылка на компонент не должна содержать имя неразбираемого компонента. На неразбираемые компоненты можно ссылаться только в значениях атрибутов, описанных как принадлежащие типу `ENTITY` или `ENTITIES`.

Ограничение по правильности: запрет на рекурсию.

Разбираемый компонент не должен содержать прямую или косвенную рекурсивную ссылку на самого себя.

Ограничение по правильности: в определении типа документа.

Ссылки на параметрические компоненты могут появляться только в определении типа документа (DTD).

Примеры ссылок на символы и на компоненты:

```
Type <key>less-than</key> (&#x3C;) to save options
This document was prepared on &docdate; and
Is classified &security-level;.
```

Примеры ссылок на параметрический компонент:

```
<!-- declare the parameter entity "ISOLat2"... -->
<!ENTITY %ISOLat2
       SYSTEM "http://www.xml.com/iso/isolat2-xml.entities" >
<!-- ... now reference it. -->
%ISOLat2;
```

4.2. Описания компонентов

Компоненты описываются следующим образом:

Описания компонентов (Entity Declaration)		
[70] EntityDecl	::=	GEDecl | PEDecl
[71] GEDecl	::=	<!ENTITY' S Name S EntityDef S? '>'
[72] PEDecl	::=	<!ENTITY' S '%' S Name S PEDef S? '>'
[73] EntityDef	::=	EntityValue | (ExternalID NDataDecl?)
[74] PEDef	::=	EntityValue | ExternalID

Значение имени (Name) идентифицирует компонент в ссылке на компонент или, в случае неразбираемого компонента, в значении атрибутов ENTITY или ENTITIES. Если один и тот же компонент описан более одного раза, то присваивается первое встреченное описание. По выбору пользователя XML-процессор может выдавать предупреждение, если дано несколько описаний компонента.

4.2.1. Внутренние компоненты

Если определение компонента – это выражение типа EntityValue (значение компонента), то определяемый компонент называется *внутренним компонентом* (internal entity). Для таких компонентов не существует отдельного физического объекта хранения, а содержание компонента дается в описании. Следует учесть, что может потребоваться некоторая обработка компонента и ссылок на символы в символьном значении компонента, чтобы создать правильный текст замены (см. раздел 4.5 «Составление текста замены внутреннего компонента»).

Внутренний компонент является разбираемым компонентом.

Пример описания внутреннего компонента:

```
<!ENTITY Pub-Status "This is a pre-release of the specification.">
```

4.2.2. Внешние компоненты

Если компонент не является внутренним, то это *внешний компонент* (external entity). Он описывается следующим образом:

Описание внешнего компонента (External Entity Declaration)

[75]	ExternalID	::=	SYSTEM' S SystemLiteral | 'PUBLIC' S PubidLitelal S SystemLiteral
[76]	NDataDecl	::=	S 'NDATA' S Name [VC: описанное обозначение]

Если присутствует выражение NDataDecl, то компонент неразбираемый, в противном случае это разбираемый компонент.

Ограничение по состоятельности: описанное обозначение.

Значение имени (Name) должно соответствовать описанному имени обозначения.

Системная константа (SystemLiteral) называется *системным идентификатором* (system identifier) компонента. Это URI, который может быть использован для нахождения компонента. Следует учесть, что знак «диез» (#) и идентификатор фрагмента, часто используемые с URI, формально не являются частью самого URI; XML-процессор может сообщить об ошибке, если идентификатор фрагмента дан как часть системного идентификатора. Пока не предоставлена другая информация вне рамок данной спецификации (например, пока не определен специальный тип элементов XML при помощи частного DTD, или команда приложения, определяемая спецификацией конкретного приложения), относительные URI соответствуют положению ресурса, в котором встретилось описание компонента. Таким образом, URI может относиться к компоненту документа, или к компоненту, содержащему внешнее подмножество DTD, или к некоторым другим параметрическим компонентам.

XML-процессор должен обрабатывать не-ASCII символ в URI, представляя этот символ в кодировке UTF-8 как один или больше байт, а затем избавляясь от

этих байт при помощи соответствующего механизма URI (то есть, конвертируя каждый байт в форму %HH, где HH представляет собой шестнадцатеричное обозначение величины байта).

В дополнение к системному идентификатору внешний идентификатор может включать в себя *общий идентификатор* (public identifier). XML-процессор, пытаясь получить содержание компонента, может использовать общий идентификатор для попытки сгенерировать альтернативный URI. Если процессору не удается это сделать, он должен использовать URI, указанный в системной константе. Перед попыткой сравнения все строки в общем идентификаторе, состоящие из пробельных литер, должны быть нормализованы к одиночному символу пробела (`#x20`), а все пробельные литеры в начале и окончании строк должны быть удалены.

Примеры описаний внешних компонентов:

```
<!ENTITY open-hatch
SYSTEM "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY open-hatch
PUBLIC "-//Textuality//TEXT Standard open-hatch boilerplate//EN"
"http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY
SYSTEM "../grafix/OpenHatch.gif"
NDATA gif >
```

4.3. Разбираемые компоненты

4.3.1. Описание текста

Каждый внешний компонент может начинаться с *описания текста* (text declaration).

Описание текста (Text Declaration)

[77] TextDecl ::= <?xml' VersionInfo? EncodingDecl S? '?>'

Описание текста должно быть представлено в символьной форме, представление в форме ссылки на разбираемый компонент не допускается. Описание текста не может появляться ни в каком месте, кроме как в начале внешнего разбираемого компонента.

4.3.2. Правильные разбираемые компоненты

Компонент документа является правильным, если он соответствует правилу вывода, обозначенному `document`. Внешний обычный разбираемый компонент является правильным, если он соответствует правилу вывода, обозначенному как `extParsedEnt`. Внешний параметрический компонент является правильным, если он соответствует правилу вывода, обозначенному как `extPE`.

Правильный внешний разбираемый компонент (Well-Formed External Parsed Entity)

[78] extParsedEnt ::= TextDecl? content

[79] extPE ::= TextDecl? extSubsetDecl

Внутренний обычный разбираемый компонент является правильным, если его текст замены соответствует правилу вывода, называемому `content`. Все внутренние

параметрические компоненты являются правильными по определению.

Из факта правильности в компонентах вытекает, что логическая и физическая структуры в XML-документе правильно вложены; ни открывающий тэг, ни закрывающий тэг, ни тэг пустого элемента, ни элемент, ни комментарий, ни команда приложения, ни ссылка на символ, ни ссылка на компонент не может начинаться в одном компоненте и заканчиваться в другом.

4.3.3. Кодировка символов в компонентах

Каждый внешний разбираемый компонент в XML-документе может использовать разную кодировку для символов. Все XML-процессоры должны уметь считывать компоненты как в кодировке UTF-8, так и в кодировке UTF-16.

Компоненты в кодировке UTF-16 должны начинаться со знака порядка байт (Byte Order Mark), описанного в Дополнении E (Annex E) стандарта ISO/IEC 10646 и в Приложении B (Appendix B) стандарта Unicode (символ неразрывного пробела нулевой ширины, ZERO WIDTH NO-BREAK SPACE character, #xFEFF). Данный знак является символом кодировки, и не является ни частью разметки, ни частью символьных данных XML-документа. XML-процессор должен уметь использовать этот символ, чтобы различать документы, написанные в кодировках UTF-8 и UTF-16.

Единственное, что требуется от XML-процессора – это считывание компонентов в кодировках UTF-8, UTF-16. Однако при этом имеется в виду, что повсеместно применяются и другие кодировки, поэтому может потребоваться, чтобы XML-процессор считывал компоненты, использующие кодировки, отличные от UTF-8 и UTF-16. Разбираемые компоненты, написанные не в кодировках, UTF-8, UTF-16 должны начинаться с описания текста, содержащего описание кодировки:

Описание кодировки (Encoding Declaration)

```
[80] EncodingDecl  ::=  S 'encoding' Eq (' " ' EncName ' " '
                        | " ' " EncName " ' " )
[81] EncName       ::=  [A-Za-z] ([A-Za-z0-9._] | '-')*      /* Название кодировки содержит
                                                              только латинские символы */
```

В компоненте документа описание кодировки является частью XML-описания. Для обозначения имени использованной кодировки применяется выражение `EncName`.

В описаниях кодировок следует использовать значения `UTF-8`, `UTF-16`, `ISO-16646-UCS-2` и `ISO-16646-UCS-4` для различных кодировок и преобразований стандартов Unicode/ISO/IEC; `ISO-8859-1`, `ISO-8859-2`, ... `ISO-8859-9` для частей стандарта ISO 8859; `ISO-2022-JP`, `Shift_JIS` и `EUC-JP` для различных форм JIS X-0208-1997. XML-процессоры могут распознавать и другие кодировки; рекомендуется, чтобы ссылки на кодировки символов, зарегистрированные в IANA как *наборы символов* (charsets), отличающиеся от перечисленных выше, осуществлялись с использованием их регистрационных имен. Подразумевается, что в этих регистрационных именах не различаются строчные и заглавные буквы, так что процессоры должны осуществлять сопоставление безотносительно к регистру.

При отсутствии информации, предоставляемой внешним транспортным протоколом (таким как HTTP или MIME) считаются ошибками предоставление XML-процессору компонента, содержащего описание кодировки:

- в кодировке, отличной от указанной в этом описании;
- появление описания кодировки не в начале внешнего компонента;
- использование кодировки, отличной от UTF-8, для компонента, который начинается не со знака порядка байт и не с описания кодировки.

Следует учесть, что поскольку ASCII является подмножеством UTF-8, компоненты в кодировке ASCII не требуют обязательного описания кодировки.

Ситуация, когда XML-процессор встречает компоненты в кодировке, которую он не способен обработать, считается неисправимой ошибкой.

Примеры описаний кодировки:

```
<?xml encoding='UTF-8'>
<?xml encoding='EUC-JP'>
```

4.4. Обработка XML-процессором компонентов и ссылок

В таблице, приведенной ниже, суммированы контекстные ситуации, в которых могут встречаться ссылки на символы, ссылки на компоненты, вызовы неразбираемых компонентов. В ней также описано требуемое поведение XML-процессора в данных ситуациях. В левом столбце приведенной ниже таблицы указаны распознаваемые контекстные ситуации.

Ссылка в содержании
как ссылка в любом месте после открывающего и до закрывающего тэга элемента; соответствует нетерминальному выражению `content`.

Ссылка в значении атрибута
как ссылка либо в значении атрибута в открывающем тэге, либо в значении по умолчанию в описании атрибута; соответствует нетерминальному выражению `AttValue`.

Встречается как значение атрибута
как значение `Name`, но не как ссылка, появляющееся либо в значении атрибута, описанного как тип `ENTITY`, либо как один из идентификаторов, разделенных пробелами, в значении атрибута, описанного как тип `ENTITIES`.

Ссылка в значении компонента
как ссылка в значении параметрического компонента или в символьном значении внутреннего компонента в описании компонента; соответствует нетерминальному выражению `EntityValue`.

Ссылка в определении типа документа (DTD)
как ссылка либо во внутреннем, либо во внешнем подмножестве DTD, но не в значениях `EntityValue` или `AttValue`.

	Тип компонента				Символ
	Параметрический	**Внутренний обычный**	**Внешний разбираемый обычный**	**Неразбираемый**	
Ссылка в содержании	Не распознается	Включается	Включается, если проводится проверка на состоятельность	Запрещается	Включается
Ссылка в значении атрибута	Не распознается	Включается в символьном виде	Запрещается	Запрещается	Включается
Встречается как значение атрибута	Не распознается	Запрещается	Запрещается	Обозначается	Не распознается
Ссылка в значении компонента	Подставляется в символьном виде	Пропускается	Пропускается	Запрещается	Подставляется
Ссылка в определении типа документа (DTD)	Подставляется как параметрический компонент	Запрещается	Запрещается	Запрещается	Запрещается

4.4.1. Не распознается

Вне пределов определения типа документа (DTD), символ `"%"` не имеет специального значения. Другими словами, то, что в DTD было бы ссылкой на параметрический компонент, в содержании не распознается в качестве разметки. Аналогично не распознаются имена неразбираемых компонентов, когда они встречаются в значении атрибута, описанного должным образом.

4.4.2. Подставляется

Компонент является *подставленным* (included), когда его текст замены получен и обработан, как если бы он был частью документа, находящейся непосредственно в том месте, где была распознана ссылка на компонент. Текст замены может содержать как символьные данные, так и (кроме параметрических компонентов) разметку, которые могут быть распознаны обычным способом; однако при этом текст замены компонентов, используемый для замены ограничителей разметки (компоненты `amp`, `lt`, `gt`, `apos`, `quot`) всегда обрабатывается как данные. (Строка `"AT&T"` превращается в `"AT&T"`, и оставшийся знак амперсанда уже не распознается как ограничитель ссылки на компонент). Ссылка на символ является *подставленной* (included), когда обработка указанного символа осуществляется непосредственно на месте самой ссылки.

4.4.3. Подставляется, если проводится верификация

Когда XML-процессор распознает ссылку на разбираемый компонент, то для того чтобы проверить состоятельность документа, процессор должен подставить на место ссылки текст замены компонента. Если компонент является внешним и процессор не пытается проверить состоятельность XML-документа, процессор может, но не обязан, подставить текст замены компонента. Если неверифицирующий анализатор не подставляет текст замены, он должен проинформировать приложение, что распознал компонент, но не прочитал его.

Это правило основано на признании того факта, что автоматическая подстановка, которую обеспечивают механизмы SGML и XML и которая изначально была предназначена для поддержания модульности при разработке, не обязательно пригодна для других приложений, в частности для просмотра документов. Например, у браузеров при обнаружении ссылки на внешний компонент есть возможность предоставлять визуальное указание на наличие компонента и получать содержание компонента для вывода на экран только по дополнительному требованию.

4.4.4. Запрещается

Запрещено и приводит к неисправимым ошибкам:

- появление ссылки на неразбираемый компонент;
- появление в определении типа документа (DTD) любого символа или ссылки на обычный компонент, кроме как в значениях `EntityValue` и `AttValue`;
- ссылка на внешний компонент в значении атрибута.

4.4.5. Подставляется в символьном виде

Когда ссылка на компонент появляется в значении атрибута или ссылка на параметрический компонент появляется в символьном значении компонента, текст замены обрабатывается на месте самой ссылки, как если бы он был частью документа, находящейся на том месте, где была распознана ссылка, за исключением того, что символы в одинарных и двойных кавычках всегда обрабатываются как нормальные символы данных и не означают конец символьной строки (литерала). Например, такая запись является правильной:

```
<!ENTITY % YN '"Yes"' >
<!ENTITY WhatHeSaid "HeSaid &YN;" >
```

а такая – нет:

```
<!ENTITY EndAttr "27'" >
<element attribute='a-&EndAttr;>
```

4.4.6. Обозначается

Когда имя неразбираемого компонента появляется в качестве идентификатора в значении атрибута, описанного типом `ENTITY` или `ENTITIES`, верифицирующий процессор должен проинформировать приложение о системном и общем (если таковой имеется) идентификаторах, как для компонента, так и для ассоциированного с ним обозначения.

4.4.7. Пропускается

Когда ссылка на обычный компонент встречается в значении `EntityValue` в описании компонента, она пропускается при разборе и сохраняется как есть.

4.4.8. Подставляется как параметрический компонент

Параметрические компоненты, так же как и внешние разбираемые компоненты, должны быть подставлены только в том случае, если проводиться проверка на состоятельность. Когда в определении типа документа (DTD) распознана и

подставлена ссылка на параметрический компонент, проводится дополнение ее текста замены за счет добавления в начало и в конец по одному символу пробела (`#x20`); это делается для того, чтобы текст замены параметрических компонентов содержал только целое число грамматических лексем в DTD.

4.5. Конструкция текста замены внутреннего компонента

При обсуждении обработки внутренних компонентов полезно различать две формы их значений. *Символьное значение компонента* (literal entity value) представляет собой строку, заключенную в кавычки, реально расположенную в описании компонента; оно соответствует терминальному значению `EntityValue`. *Текст замены* (replacement text) представляет собой содержание компонента после замены ссылок на символы и ссылок на параметрические компоненты.

Символьное значение компонента, как это дано в описании внутреннего компонента (`EntityValue`), может содержать ссылки на символы, параметрические компоненты и обычные компоненты. Подобные ссылки должны полностью содержаться в символьном значении компонента. Реальный текст замены, который подставляется описанным выше способом, должен содержать *текст замены* любого относящегося к нему параметрического компонента, а также должен содержать символ, на который имелась ссылка, на месте любой ссылки на символ в символьном значении компонента. Однако ссылки на обычные компоненты должны быть оставлены в том виде, в каком они были распознаны, то есть неразвернутыми. Например, даны следующие описания:

```
<!ENTITY % pub     "&#xc9;Editions Gallimard" >
<!ENTITY   rights  "All rights reserved" >
<!ENTITY   book        "La Peste: Albert Camus,
&#xc9; 1947 %pub;. &rights;" >
```

тогда текст замены для компонента `"book"` будет выглядеть так:

```
La Peste: Albert Camus,
© 1947 Editions Gallimard. &rights
```

Ссылка на обычный компонент `"&right;"` была бы раскрыта, если бы в содержании документа или в значении атрибута появилась ссылка `"&book;"`.

Эти простые правила могут иметь сложную взаимосвязь; подробное рассмотрение более сложного примера дано в дополнении D «Раскрытие ссылок на компоненты и символы».

4.6. Предопределенные компоненты

Как ссылки на компоненты, так и ссылки на символы можно использовать, чтобы *обойти* (escape) необходимость написания левой угловой скобки, знака амперсанда и других ограничителей. Для этих целей существует набор обычных компонентов (`amp`, `lt`, `gt`, `apos`, `quot`). Можно также использовать численные ссылки на символы; эти ссылки раскрываются сразу, как только распознаются. Обрабатываться они должны как символьные данные, так что численные ссылки на символы `"&#60;"` и `"&#38;"` могут быть использованы для обхода написания

левой угловой скобки (<) и знака амперсанда (&), когда эти знаки встречаются в символьных данных.

Все XML-процессоры должны распознавать эти предопределенные компоненты вне зависимости от того, описаны они или нет. Для взаимодействия в состоятельных XML-документах эти компоненты перед использованием должны быть описаны так же, как и все остальные. Если рассматриваемые компоненты описаны, они должны быть описаны как внутренние компоненты; текст их замены состоит из одиночного символа, непосредственное использование которого необходимо обойти, или из ссылки на данный символ, как показано в примерах.

```
<!ENTITY   lt      "&#38;&#60;">
<!ENTITY   gt      "&#62;">
<!ENTITY   amp     "&#38;&#38;">
<!ENTITY   apos    "&#39;;">
<!ENTITY   quot    "&#34;">
```

Следует заметить: чтобы соблюсти требование правильности замены компонента, символы < е & в описаниях "lt" е "amp" обходятся дважды.

4.7. Описания обозначений

Обозначения (notations) идентифицируют по имени формат неразбираемых компонентов, формат элементов, несущих атрибут обозначения, или приложение, которому адресованы команды приложения.

Описания обозначений (notation declarations) предоставляют имя обозначения (нотацию) для использования в описаниях компонентов и списков атрибутов и в спецификациях атрибутов, а также внешний идентификатор обозначения, который может позволить XML-процессору или клиентскому приложению определить расположение вспомогательной программы, способной обрабатывать данные в указанном обозначении.

Описания обозначений (notation declarations)

```
[82] NotationDecl ::= '<!NOTATION' S Name S (ExternalID | PublicID) S? '>'
[83] PublicID     ::= 'PUBLIC' S PubidLiteral
```

XML-процессоры должны предоставить приложениям имя и внешний (внешние) идентификатор (идентификаторы) любого описанного обозначения, а также любого другого, на которое имеется ссылка в значении атрибута, в описании атрибута или в описании компонента. Дополнительно они могут разложить внешний идентификатор на системный идентификатор, имя файла или другую информацию, необходимую, чтобы позволить приложению запросить у процессора данные из описанного обозначения. (Однако не считается ошибкой, если XML-документ описывает и ссылается на обозначение, для которого в системе, где работают XML-процессор и приложение, не существует специфического приложения.)

4.8. Компонент документа

Компонент документа (document entity) представляет собой корень дерева компонентов и отправную точку для XML-процессора. Данная спецификация

не указывает, каким образом XML-процессор должен обнаруживать компонент документа; в отличие от других компонентов, компонент документа не имеет имени и может появиться в потоке ввода процессора вообще без какого-либо обозначения.

5. Соответствие

5.1. Верифицирующие и неверифицирующие процессоры

Соответствующие XML-процессоры делятся на два класса: верифицирующие и неверифицирующие.

Верифицирующие и неверифицирующие процессоры одинаково должны сообщать о нарушениях ограничений по правильности в содержании компонента документа и в любом другом считываемом ими разбираемом компоненте.

Верифицирующие процессоры (validating processors) должны сообщать о нарушениях ограничений, определенных в описаниях, находящихся в определении типа документа (DTD), и о неспособности удовлетворить ограничения по состоятельности, описанные в данной спецификации. Чтобы выполнить это требование, верифицирующие XML-процессоры должны полностью считывать и обрабатывать определение типа документа, а также считывать и обрабатывать все внешние разбираемые компоненты, упомянутые в документе.

Неверифицирующие процессоры должны проверять на правильность только компонент документа, включая все внутреннее подмножество DTD. Несмотря на то, что от них не требуется проверять документ на состоятельность, не проверяющие на состоятельность процессоры должны *обрабатывать* (process) все описания, считываемые из внутреннего подмножества DTD и из любого параметрического компонента, вплоть до первой ссылки на параметрический компонент, которую они *не* считывают. Другими словами, они должны использовать информацию из считанных описаний для нормализации значений атрибутов, подстановки текста замены внутренних компонентов и предоставления значений атрибутов по умолчанию. Они не должны обрабатывать описания компонентов или описания списков атрибутов, которые встретились после ссылки на не считываемый ими параметрический компонент, поскольку в компоненте могут находиться описания, переопределяющие некоторые величины.

5.2. Использование XML-процессоров

Поведение верифицирующего XML-процессора в значительной степени предсказуемо. Он должен прочитать все части документа и сообщить о нарушениях ограничений по правильности и по состоятельности. Неверифицирующий процессор выполняет более узкую задачу; ему не надо считывать никаких частей документа кроме компонента документа. Этим обусловлены два обстоятельства, которые могут иметь значение для пользователей XML-процессоров:

- некоторые ошибки правильности, особенно требующие чтения внешних компонентов, могут быть не обнаружены неверифицирующим процессором. Это относится к ограничениям, называемым «описанный компонент», «разбираемый компонент», «запрет на рекурсию», а также к некоторым случаям,

описанным как «запрещается» в разделе 4.4 «Обработка XML-процессором компонентов и ссылок»;

- информация, передаваемая процессором приложению, может различаться в зависимости от того, осуществляет ли процессор считывание внешних и параметрических компонентов. Например, неверифицирующий процессор может не нормализовать значения атрибутов, не подставлять текст замены внутреннего компонента или не предоставлять значения атрибутов по умолчанию, если выполнение этих действий зависит от считывания внешних компонентов и параметрических компонентов.

Чтобы различные XML-процессоры взаимодействовали максимально надежно, приложения, использующие неверифицирующие процессоры, не должны полагаться на какое-либо необязательное действие таких процессоров. Приложения, которым необходимы такие возможности, как использование атрибутов по умолчанию или использование внутренних компонентов, описанных во внешних компонентах, – должны применять верифицирующие XML-процессоры.

6. Нотация

Формальная грамматика языка XML дается в данной спецификации с использованием простой расширенной формы обозначений Бэкуса-Наура (Extended Backus-Naur Form, EBNF). Каждое правило в грамматике дает определение одного символа в виде:

```
symbol ::= expression
```

Если символы определены регулярным выражением, то они записываются с заглавной буквы, в противном случае со строчной. Символьные строки заключаются в кавычки.

В правой части правил вместо строк, состоящих из одного или более символов, используется следующее выражение:

```
#xN
```

где `N` представляет собой шестнадцатеричное число; выражение соответствует символу в стандарте ISO/IEC 10646, каноническое значение кода которого (UCS-4) при его интерпретации как двоичного числа без знака имеет указанное значение `N`. В форме `#xN` не важно, сколько нулей стоит в начале; число нулей в начале соответствующего значения кода зависит от используемой кодировки символов и в XML не существенно.

```
[A-zA-Z], [#xN-#xN]
```

соответствует любому символу в указанном диапазоне (диапазонах), включая границы.

```
[^a-z] [^#xN-#xN]
```

соответствует любому символу *вне* указанного диапазона (указанных диапазонов).

```
[^abc], [^#xN#xN#xN]
```

соответствует любому символу со значением, не находящимся среди указанных символов.

```
"string"
```

представляет собой буквенную строку, соответствующую выражению, находящемуся между двойными кавычками.

```
'string'
```

представляет собой буквенную строку, соответствующую выражению, находящемуся между одинарными кавычками.

Эти символы можно комбинировать для составления более сложных конструкций, как показано в следующих примерах, где `A` и `B` представляют одиночные выражения:

```
(expression)
```

где `(expression)` рассматривается как некая целая единица, оно может быть сгруппировано, как это описано ниже:

```
A?
```

соответствует выражению `A` или ничему; необязательному `A`;

```
A B
```

соответствует выражению `A`, за которым следует выражение `B`;

```
A | B
```

соответствует выражению `A` или выражению `B`, но не обоим;

```
A-B
```

это любая строка, соответствующая выражению `A`, но не соответствующая выражению `B`;

```
A+
```

соответствует одному или более повторений выражения `A`;

```
A*
```

соответствует повторению выражения A ноль или более раз.

Другие обозначения, использованные в данном представлении, таковы:

```
/* ... */
```

комментарий;

```
[wfc: ... ]
```

ограничение по правильности. Имя, следующее после двоеточия, указывает на ограничение по правильности, налагаемое на документы, связанные с правилом вывода;

```
[vc: ... ]
```

ограничение по состоятельности. Имя, следующее после двоеточия, указывает ограничение по состоятельности, налагаемое на документы, связанные с представлением.

Дополнения

A. Литература

A.1. Нормативная литература

IANA

(Internet Assigned Numbers Authority) *Official Names for Character Sets,* ed. Keld Simonsen et al. See ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets.

IETF RFC 1766

IETF (Internet Engineering Task Force). RFC 1766: *Tags for the Identifications of Languages*, ed. H. Alvestrand. 1995.

ISO 639

(International Organization for Standardization). *ISO 639:1988 (E). Code for the representation of names of languages.* [Geneva]: International Organization for Standardization, 1988.

ISO 3166

(International Organization for Standardization). *ISO 3166-1:1997 (E). Codes for the representation of names of countries and their subdivisions – Part1: Country codes* [Geneva]: International Organization for Standardization, 1997.

ISO/IEC 10646

ISO (International Organization for Standardization). *ISO/IEC 10646-1993 (E). Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basis Multilingual Plane.* [Geneva]: International Organization for Standardization, 1993 (plus amendments AM 1 through AM 7).

Unicode

The Unicode Consortium. *The Unicode Standard, Version 2.0*. Reading, Mass.: Addison-Wesley Developers Press, 1996.

A.2. Другая литература

Aho/Ullman

Aho, Alfred V., Ravi Sethi, and Jeffery D. Ullman. *Compilers: Principles, Techniques, and Tools*. Reading: Addison-Wesley, 1986, rpt. corr. 1988.

Berners-Lee et al.

Berners-Lee, T., R. Fielding, and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax and Semantics*. 1997. (Work in progress; see updates to RFC1738.)

Brüggemann-Klein

Brüggemann-Klein, Anne. *Regular Expressions into Finite Automata*. Extended abstract in I. Simon, Hrsg., LATIN 1992, S. 97-98. Springer-Verlag, Berlin 1992. Full version in Theoretical Computer Science 120: 197-213. 1993.

Brüggemann-Klein and Wood

Brüggemann-Klein, Anne, and Derick Wood. *Deterministic Regular Languages*. Universität Freiburg, Institut für Informatik, Bericht 38, Oktober 1991.

Clark

James Clark. Comparison of SGML and XML. See http://www.w3.org/TR/NOTE-sgml-xml-971215.

IETF RFC1738

IETF (Internet Engineering Task Force). *RFC 1738: Uniform Resource Locators (URL)*, ed. T. Berners-Lee, L. Masinter, M. McCahill. 1994.

IETF RFC1808

IETF (Internet Engineering Task Force). *RFC 1808: Relative Uniform Resource Locators (URL)*, ed. R. Fielding. 1995.

IETF RFC2141

IETF (Internet Engineering Task Force). *RFC 2141: URN Syntax*, ed. R. Moats. 1997.

ISO 8879

ISO (International Organization for Standardization). *ISO 8879:1986(E). Information processing – Text and Office Systems – Standard Generalized Markup Language (SGML)*. First edition – 1986-10-15. [Geneva]: International Organization for Standardization, 1986.

ISO/IEC 10744

ISO (International Organization for Standardization). *ISO/IEC 10744-1992 (E). Information technology – Hypermedia/Time-based Structuring Language (HyTime)*. [Geneva]: International Organization for Standardization, 1992. *Extended Facilities Annexe*. [Geneva]: International Organization for Standardization, 1996.

B. Классы символов

Согласно характеристикам, определенным в стандарте Unicode, символы классифицируются как основные (BaseChar) символы (среди прочих они содержат алфавитные символы латинского алфавита без диакритических знаков), идеографические (ideographic) и комбинированные (combining) символы (этот класс содержит большинство диакритических символов); класс букв образуется в результате объединение данных классов. Различают также цифры (digit) и расширители (extender).

Символы (Characters)		
[84] Letter	::=	BaseChar | Ideographic
[85] BaseChar	::=	[#0041x-#x005A] | [#x0061-#x007A] | [#x00C0-#x00D6] | [#x00D8-#x00F6] | [#x00F8-#x00FF] | [#x0100-#x0131] | [#x0134-#x013E] | [#x0141-#x0148] | [#x014A-#x017E] | [#x0180-#x01C3] | [#x01CD-#x01F0] | [#x01F4-#x01F5] | [#x01FA-#x0217] | [#x0250-#x02A8] | [#x02BB-#x02C1] | #x0386 | [#x0388-#x038A] | #x038C | [#x038E-#x03A1] | [#x03A3-#x03CE] | [#x03D0-#x03D6]

Символы (Characters)

| #x03DA | [#x03DC-#x03DE] | #x03E0 | [#x03E2-#x03F3] | [#x0401-#x040C] | [#x040E-#x044F] | [#x0451-#x045C] | [#x045E-#x0481] | [#x0490-#x04C4] | [#x04C7-#x04C8] | [#x04CB-#x04CC] | [#x04D0-#x04EB] | [#x04EE-#x04F5] | [#x04F8-#x04F9] | [#x0531-#x0556] | #x0559 | [#x0561-#x0586] | [#x05D0-#x05EA] | [#x05F0-#x05F2] | [#x0621-#x063A] | [#x0641-#x064A] | [#x0671-#x06B7] | [#x06BA-#x06BE] | [#x06C0-#x06CE] | [#x06D0-#x06D3] | #x06D5 | [#x06E5-#x06E6] | [#x0905-#x0939] | #x093D | [#x0958-#x0961] | [#x0985-#x098C] | [#x098F-#x0990] | [#x0993-#x09A8] | [#x09AA-#x09B0] | #x09B2 | [#x09B6-#x09B9] | [#x09DC-#x09DD] | [#x09DF-#x09E1] | [#x09F0-#x09F1] | [#x0A05-#x0A0A] | [#x0A0F-#x0A10] | [#x0A13-#x0A28] | [#x0A2A-#x0A30] | [#x0A32-#x0A33] | [#x0A35-#x0A36] | [#x0A38-#x0A39] | [#x0A59-#x0A5C] | #x0A5E | [#x0A72-#x0A74] | [#x0A85-#x0A8B] | #x0A8D | [#x0A8F-#x0A91] | [#x0A93-#x0AA8] | [#x0AAA-#x0AB0] | [#x0AB2-#x0AB3] | [#x0AB5-#x0AB9] | #x0ABD | #x0AE0 | [#x0B05-#x0B0C] | [#x0B0F-#x0B10] | [#x0B13-#x0B28] | [#x0B2A-#x0B30] | [#x0B32-#x0B33] | [#x0B36-#x0B39] | #x0B3D | [#x0B5C-#x0B5D] | [#x0B5F-#x0B61] | [#x0B85-#x0B8A] | [#x0B8E-#x0B90] | [#x0B92-#x0B95] | [#x0B99-#x0B9A] | #x0B9C | [#x0B9E-#x0B9F] | [#x0BA3-#x0BA4] | [#x0BA8-#x0BAA] | [#x0BAE-#x0BB5] | [#x0BB7-#x0BB9] | [#x0C05-#x0C0C] | [#x0C0E-#x0C10] | [#x0C12-#x0C28] | [#x0C2A-#x0C33] | [#x0C35-#x0C39] | [#x0C60-#x0C61] | [#x0C85-#x0C8C] | [#x0C8E-#x0C90] | [#x0C92-#x0CA8] | [#x0CAA-#x0CB3] | [#x0CB5-#x0CB9] | #x0CDE | [#x0CE0-#x0CE1] | [#x0D05-#x0D0C] | [#x0D0E-#x0D10] | [#x0D12-#x0D28] | [#x0D2A-#x0D39] | [#x0D60-#x0D61] | [#x0E01-#x0E2E] | #x0E30 | [#x0E32-#x0E33] | [#x0E40-#x0E45] | [#x0E81-#x0E82] | #x0E84 | [#x0E87-#x0E88] | #x0E8A | #x0E8D | [#x0E94-#x0E97] | [#x0E99-#x0E9F] | [#x0EA1-#x0EA3] | #x0EA5 | #x0EA7 | [#x0EAA-#x0EAB] | [#x0EAD-#x0EAE] | #x0EB0 | [#x0EB2-#x0EB3] | #x0EBD | [#x0EC0-#x0EC4] | [#x0F40-#x0F47] | [#x0F49-#x0F69] | [#x10A0-#x10C5] | [#x10D0-#x10F6] | #x1100 | [#x1102-#x1103] | [#x1105-#x1107] | #x1109 | [#x110B-#x110C] | [#x110E-#x1112] | #x113C | #x113E | #x1140 | #x114C | #x114E | #x1150 | [#x1154-#x1155] | #x1159 | [#x115F-#x1161] | #x1163 | #x1165 | #x1167 | #x1169 | [#x116D-#x116E] | [#x1172-#x1173] | #x1175 | #x119E | #x11A8 | #x11AB | [#x11AE-#x11AF] | [#x11B7-#x11B8] | #x11BA | [#x11BC-#x11C2] | #x11EB | #x11F0 | #x11F9 | [#x1E00-#x1E9B] | [#x1EA0-#x1EF9] | [#x1F00-#x1F15] | [#x1F18-#x1F1D] | [#x1F20-#x1F45] | [#x1F48-#x1F4D] | [#x1F50-#x1F57] | #x1F59 | #x1F5B | #x1F5D | [#x1F5F-#x1F7D] | [#x1F80-#x1FB4] | [#x1FB6-#x1FBC] | #x1FBE | [#x1FC2-#x1FC4] | [#x1FC6-#x1FCC] | [#x1FDO-#x1FD3] | [#x1FD6-#x1FDB] | [#x1FE0-#x1FEC] | [#x1FF2-#x1FF4] | [#x1FF6-#x1FFC] | #x2126 | [#x3041-#x3094] | [#x30A1-#x30FA] | [#x3105-#x312C] | [#xAC00-#xD7A3]

[86] Ideographic ::= [#x4E00-#x9FA5] | #x3007 | [#x3021-#x3029]

[87] CombiningChar ::= [#x0300-#x0345] | [#x0360-#x0361] | [#x0483-#x0486] | [#x0591-#x05A1] | [#x05A3-#x05B9] | [#x05BB-#x05BD] | #x05BF | [#x05C1-#x05C2] | #x05C4 | [#x064B-#x0652] | #x0670 | [#x06D6-#x06DC] | [#x06DD-#x06DF] | [#x06E0-#x06E4] | [#x06E7-#x06E8] | [#x06EA-#x06ED] | [#x0901-#x0903] | #x093C [#x093E-#x094C] | #x094D | [#x0951-#x0954] | [#x0962-#x0963] | [#x0981-#x0983] | #x09BC | #x09BE | #x09BF | [#x09C0-#x09C4] | [#x09C7-#x09C8] | [#x09CB-#x09CD] | #x09D7 | [#x09E2-#x09E3] | #x0A02 | #x0A3C | #x0A3E | #x0A3F | [#x0A40-#x0A42] | [#x0A47-#x0A48] | [#x0A4B-#x0A4D] | [#x0A70-#x0A71] | [#x0A81-#x0A83] | #x0ABC | [#x0ABE-#x0AC5] | [#x0AC7-#x0AC9] | [#x0ACB-#x0ACD] | [#x0B01-#x0B03] | #x0B3C | [#x0B3E-#x0B43] | [#x0B47-#x0B48] | [#x0B4B-#x0B4D] | [#x0B56-#x0B57] | [#x0B82-#x0B83] | [#x0BBE-#x0BC2] | [#x0BC6-#x0BC8] | [#x0BCA-#x0BCD] | #x0BD7 | [#x0C01-#x0C03] | [#x0C3E-#x0C44] | [#x0C46-#x0C48] | [#x0C4A-#x0C4D] | [#x0C55-#x0C56] | [#x0C82-#x0C83] | [#x0CBE-#x0CC4] | [#x0CC6-#x0CC8] | [#x0CCA-#x0CCD]

Символы (Characters)

	[#x0CD5-#x0CD6]	[#x0D02-#x0D03]	[#x0D3E-#x0D43]	[#x0D46-#x0D48]	[#x0D4A-#x0D4D]	#x0D57	#x0E31	[#x0E34-#x0E3A]	[#x0E47-#x0E4E]	#x0EB1	[#x0EB4-#x0EB9]	[#x0EBB-#x0EBC]	[#x0EC8-#x0ECD]	[#x0F18-#x0F19]	#x0F35	#x0F37	#x0F39	#x0F3E	#x0F3F	[#x0F71-#x0F84]	[#x0F86-#x0F8B]	[#x0F90-#x0F95]	#x0F97	[#x0F99-#x0FAD]	[#x0FB1-#x0FB7]	#x0FB9	[#x20D0-#x20DC]	#x20E1	[#x302A-#x302F]	#x3099	#x309A
[88] Digit	::= [#x0030-#x0039]	[#x0660-#x0669]	[#x06F0-#x06F9]	[#x0966-#x096F]	[#x09E6-#x09EF]	[#x0A66-#x0A6F]	[#x0AE6-#x0AEF]	[#x0B66-#x0B6F]	[#x0BE7-#x0BEF]	[#x0C66-#x0C6F]	[#x0CE6-#x0CEF]	[#x0D66-#x0D6F]	[#x0E50-#x0E59]	[#x0ED0-#x0ED9]	[#x0F20-#x0F29]
[89] Extender	::= #x00B7	#x02D0	#x02D1	#x0387	#x0640	#x0E46	#x0EC6	#x3005	[#x3031-#x3035]	[#x309D-#x309E]	[#x30FC-#x30FE]

Классы символов, определенные здесь, могут быть выведены из базы данных символов Unicode следующим образом:

- символы начала имени должны иметь один из компонентов: `Ll`, `Lu`, `Lo`, `Lt`, `Nl`;
- символы имени, отличные от символов начала имени, должны иметь один из компонентов: `Mc`, `Me`, `Mn`, `Lm` или `Nd`;
- в именах XML не разрешено использование символов из зоны совместимости (то есть имеющих код символа больше чем `#xF900` и меньше чем `#xFFFE`);
- не разрешено использование символов, имеющих шрифтовое разбиение или разбиение для совместимости (то есть имеющих *форматирующий тэг совместимости* (compatibility formatting tag) в поле 5 базы данных — помеченных полем 5, начинающимся с `"<"`);
- символы `[#x02BB-#x02C1]`, `#x0559`, `#x06E5`, `#x06E6` обрабатываются как символы начала имени, а не как символы имени, потому что файл свойств классифицирует их как алфавитные символы;
- исключаются символы `#x20DD-#x20E0` (в соответствии с разделом 5.14 стандарта Unicode);
- символ `#x00B7` классифицируется как расширитель, поскольку список свойств идентифицирует его именно так;
- символ `#x0387` добавлен как символ имени, поскольку символ `#x00B7` является его каноническим эквивалентом;
- разрешено использование символов `":"` и `"_"` в качестве символов начала имени;
- разрешено использование символов `"."` и `"-"` в качестве символов имени.

C. XML и SGML. Ненормативно

Язык XML был задуман как подмножество языка SGML, так что всякий состоятельный XML-документ должен одновременно являться и SGML-совместимым документом. Детальное сравнение дополнительных ограничений, налагаемых языком XML в дополнение к ограничениям, наложенным языком SGML, можно найти в работе Кларка (Clark).

D. Раскрытие ссылок на компоненты и символы. Ненормативно

Это дополнение содержит примеры, иллюстрирующие последовательность распознавания и раскрытия ссылок на компоненты и символы, как это указано в разделе 4.4 «Обработка XML-процессором компонентов и ссылок».

Если определение типа документа (DTD) содержит описание

```
<!ENTITY example "<p>An ampersand (&#38;#38;) may be escaped
numerically (&#38;#38;#38;) or with a general entity
```

то XML-процессор распознает ссылки на символы в процессе разбора описания компонента и раскрывает их, прежде чем сохранить последующие строки в качестве значения компонента "example":

```
<p>An ampersand (&#38;) may be escaped
numerically (&#38;#38;) or with a general entity
(&amp;amp;).</p>
```

В документе ссылка на "&example;" приведет к повторной обработке текста, во время которой будут распознаны открывающий и закрывающий тэг элемента "p", а также будут распознаны и раскрыты три ссылки. В результате появится элемент "p" со следующим содержанием (полностью состоящим из данных, без разделителей или разметки):

```
An ampersand (&) may be escaped
numerically(&#38;) or with a general entity
(&amp;).
```

Более сложный пример полностью иллюстрирует правила и вытекающие из них результаты. В этом примере строки пронумерованы исключительно для удобства дальнейшего обсуждения:

```
1 <?xml version='1.0'?>
2 <!DOCTYPE test [
3 <!ELEMENT test (#PCDATA) >
4 <!ENTITY % xx '&#37;zz;'>
5 <!ENTITY % zz '&#60; !ENTITY tricky "error-prone" >' >
6 %xx;
7 ]>
8 <test>This sample shows a &tricky; method.</test>
```

В результате получается следующее:

- в строке 4. Ссылка на символ 37 раскрывается немедленно, и параметрический компонент "xx" помещается в таблицу символов, имея значение "%zz;". Поскольку текст замены повторно не просматривается, ссылка на параметрический компонент "zz" не распознается. (Ее распознание привело бы к ошибке, поскольку компонент "zz" еще не описан.);
- в строке 5. Ссылка на символ "<" раскрывается немедленно, и параметрический компонент "zz" сохраняется вместе со вставляемым текстом "<!ENTITY tricky "error-prone" >", который является правильным описанием компонента;

- в строке 6. Распознается ссылка на параметрический компонент "xx", и происходит обработка вставляемого текста (а именно "%zz;"). Ссылка на компонент "zz" в свою очередь распознается, и проводится обработка ее текста замены ("<!ENTITY tricky "error-prone" >"). Обычный компонент "tricky" со вставляемым текстом "error-prone" не был описан;
- в строке 8. Распознается ссылка на обычный компонент "tricky", она расширяется, так что полное содержание элемента "test" является самоописывающейся строкой *This sample shows a error-prone method.* (Этот пример иллюстрирует метод, чреватый ошибками.)

E. Детерминированные модели содержания. Ненормативно

Для совместимости необходимо, чтобы модели содержания в описаниях типа элемента были детерминированными. SGML требует детерминированных моделей содержания (там они называются «недвусмысленными», (unambiguous)); XML-процессоры, построенные на SGML-системах, могут пометить недетерминированные модели как ошибочные.

Например, модель содержания ((b,c) | (b,d)) является недетерминированной, поскольку анализатор, получивший начальное выражение b, не может знать, какое из выражений b в модели будет находиться в соответствии, не заглянув вперед, чтобы посмотреть, какой элемент следует за b. В данном примере две ссылки на выражение b можно заменить на одну, причем модель содержания примет вид (b,(c | d)). Теперь начальное выражение b однозначно соответствует только одному имени в модели содержания. Анализатору не требуется заглядывать вперед, чтобы посмотреть, какой элемент идет следом; принято будет либо выражение c либо d.

Более формальный механизм: из модели содержания, используя стандартные алгоритмы, например алгоритм 3.5 из раздела 3.9 публикации Aho, Sethi and Ullman [Aho/Ullman], можно сконструировать конечный автомат. Во многих подобных алгоритмах для каждой позиции регулярного выражения создается набор элементов, следующих за данной позицией (то есть для каждого листа в синтаксическом дереве регулярного выражения); если какая-либо позиция имеет набор, в котором более чем одна из следующих позиций обозначена одним и тем же именем типа элемента, то модель содержания неверна, и о ней можно выдавать сообщение как об ошибке.

Существуют алгоритмы, позволяющие свести многие, но не все, недетерминированные модели содержания к эквивалентным детерминированным моделям (см. Брюгманн-Клейн (Brüggemann-Klein) 1991 [Брюгманн-Клейн]).

F. Автоматическое определение кодировки символов. Ненормативно

Описание кодировки XML действует как внутренняя метка на каждом компоненте, указывая, какая кодировка символов используется. Однако прежде чем XML-процессор сможет прочитать внутреннюю метку, ему, очевидно, необходимо знать используемую кодировку символов, а как раз эту информацию и должна указать внутренняя метка. В общем случае ситуация безнадежна. Однако в XML

абсолютно безнадежной ее назвать нельзя, поскольку в языке XML существуют два ограничения: предполагается, что каждая реализация поддерживает только конечный набор кодировок символов и что местоположение и содержание описания кодировки строго зафиксированы, чтобы можно было в нормальных случаях автоматически определять кодировку символов, используемую в каждом компоненте. Кроме того, во многих случаях дополнительно к самому потоку данных XML существуют и другие источники информации. Можно различать две ситуации, в зависимости от того, передается ли процессору компонент XML с сопровождающей (внешней) информацией или без нее. Для начала рассмотрим первую ситуацию.

Поскольку каждый компонент XML, написанный в формате, отличном от UTF-8 или UTF-16, *должен* начинаться с описания кодировки XML, в которой первыми символами должны быть `"<?xml"`, любой соответствующий процессор после ввода двух или четырех байт может выяснить, какой из ниже перечисленных случаев следует применять. При чтении данного списка полезно знать, что в формате UCS-4 знак `"<"` представляется как `"#x0000003C"`, знак `"?"` представляется как `"#x0000003F"`, а знак порядка байтов, требуемый в потоке данных UTF-16, представляется как `"#xFEFF"`:

- `00 00 00 3C`: UCS-4, машина с адресацией к старшему байту (big-endian), (порядок 1234);
- `3C 00 00 00`: UCS-4, машина с адресацией к младшему байту (little-endian) (порядок 4321);
- `00 00 3C 00`: UCS-4, необычный порядок байтов (2143);
- `00 3C 00 00`: UCS-4, необычный порядок байтов (3412);
- `FE FF`: UTF-16, машина с адресацией к старшему байту;
- `FF FE`: UTF-16, машина с адресацией к младшему байту;
- `00 3C 00 3F`: UTF-16, машина с адресацией к старшему байту, нет знака порядка байтов (значит, строго говоря, этот случай ошибочен);
- `00 3C 3F 00`: UTF-16, машина с адресацией к младшему байту, нет знака порядка байтов (значит, строго говоря, этот случай ошибочен);
- `3C 3F 78 6D`: кодировки UTF-8, ISO 646, ASCII, некоторые части ISO 8859, Shift-JIS, EUC или любые другие 7-битные, 8-битные кодировки или кодировки со смешанной шириной, которые гарантируют, что символы ASCII имеют нормальные положения, значения и ширину; необходимо провести считывание описания действующей кодировки, чтобы определить, что из вышеперечисленного следует использовать, но поскольку все эти кодировки применяют одинаковый битовый шаблон для ASCII символов, можно осуществить достоверное считывание самого описания кодировки;
- `4C 6F A7 94`: EBCDIC (в нескольких разновидностях; необходимо прочитать полное описание кодировки, чтобы определить, какая кодовая страница используется);
- другие: UTF-8 без описания кодировки, либо поток данных искажен, либо фрагментарен, либо информация находится в некоторой упаковке.

Данный уровень автоматического определения кодировки вполне достаточен для считывания описания кодировки XML и для анализа идентификатора кодировки

символов, который все еще необходим для того, чтобы распознать отдельные члены каждого семейства кодировок (например, отличить кодировку UTF-8 от 8859, и части кодировки 8859 друг от друга или различить конкретную используемую кодовую страницу EBCDIC и т.д.).

Поскольку в содержании описания кодировки могут применяться только ASCII-символы, процессор может гарантированно прочитать все описание кодировки, как только он определит используемую разновидность кодировок. Так как на практике все широко используемые кодировки относятся к одной из вышеперечисленных категорий, описание кодировки XML позволяет сделать обозначение кодировок символов достаточно достоверным, даже если внешние источники информации на уровне операционной системы или транспортного протокола ненадежны.

Как только процессор распознал используемую кодировку символов, он может действовать в соответствии с ситуацией, либо запуская отдельные входные процедуры для каждого случая, либо обрабатывая символы ввода соответствующими функциями преобразования.

Подобно другим самообозначающимся системам, описание кодировки XML не подействует, если какое-либо программное приложение изменит набор символов компонента или кодировку, не обновив одновременно описание кодировки. При реализации алгоритмов кодировки символов требуется тщательность, чтобы гарантировать точность внутренней и внешней информации, используемой для обозначения компонента.

Вторая возможная ситуация заключается в том, что компонент XML сопровождается информацией о кодировке, как в некоторых файловых системах и сетевых протоколах. Когда существует несколько источников информации, их относительный приоритет и предпочтительный метод разрешения конфликтов должны быть указаны как часть протокола более высокого уровня, используемого для доставки XML. Например, правила для относительного приоритета внутренних меток и меток MIME-типа во внешних заголовках, должны составлять часть RFC-документа, определяющего MIME-типы text/xml и application/xml. Однако в интересах обеспечения взаимодействия рекомендованы следующие правила:

- если компонент XML расположен в файле, то для определения кодировки символов используются знак порядка байтов и описания кодировки команды приложения (если они наличествуют); все другие эвристики и источники информации используются исключительно для устранения ошибок;
- если компонент XML предоставлен вместе с MIME-типом текста (text/xml), то метод кодировки символов определяется параметром `charset` MIME-типа; все другие эвристики и источники информации используются исключительно для устранения ошибок;
- если компонент XML предоставлен вместе с MIME-типом приложения (application/xml), то для определения кодировки символов применяются знак порядка байтов и описания кодировки команд приложения (если они наличествуют); все другие эвристики и источники информации используются исключительно для устранения ошибок.

Эти правила действуют только в случае отсутствия документации уровня протокола; в частности, когда определены MIME типы text/xml и application/xml, эти правила заменяются рекомендациями соответствующего RFC.

G. Рабочая группа по языку XML консорциума W3C. Ненормативно

Данная спецификация была подготовлена и одобрена для публикации Рабочей группой (Working Group, WG) по языку XML консорциума W3C. Одобрение данной спецификации рабочей группой не обязательно подразумевает, что все члены группы проголосовали за ее принятие. В прошлом и в настоящем в рабочую группу входили:

Йон Босак (Jon Bosak), председатель, Sun; Джеймс Кларк (James Clark), технический руководитель; Тим Брей (Tim Bray), редактор XML, Textuality and Netscape; Жан Паоли (Jean Paoli), редактор XML, Microsoft; К.М. Сперберг-МакКуин (C.M. Sperberg-MacQueen), редактор XML, U. of Ill.; Дэн Коннолли (Dan Connolly), ответственный по связям с W3C, W3C; Паула Ангерштайн (Paula Angerstein), Texcel; Стив ДеРоуз (Steve DeRose), INSO; Дейв Холландер (Dave Hollander), HP; Эллиот Кимбер (Elliot Kimber), ISOGEN; Ева Малер (Eve Mahler), ArborText; Том Маглайери (Tom Magliery), NCSA; Мюррей Мэлони (Murray Maloney), Muzmo and Grif; Макото Мурата (Makoto Murata), Fuji Xerox Information Systems; Джоэль Нава (Joel Nava), Adobe; Конлет О'Коннелл (Conleth O'Connell), Vignette; Питер Шарп (Peter Sharpe), SoftQuad; Джон Тиг (John Tigue), DataChannel.

Приложение D
XML-данные и типы данных DTD

Эта таблица скопирована из сообщения консорциума W3C, которое можно найти по адресу:

http://www.w3.org/TR/1998/NOTE-XML-data

Специальные типы данных

В табл. D.1 включены все наиболее широко используемые типы и все встроенные типы наиболее популярных баз данных, языков программирования и систем, таких как SQL, Visual Basic, C, C++ и Java™.

Таблица D.1

Имя	Тип в разборе	Тип в хранении	Примеры
string	pcdata	Строка (string, Unicode)	*Это строка Unicode*
number	Число без ограничений на количество цифр, может иметь знак, цифры дробной части и необязательный показатель степени. Пунктуация как в американском английском	Строка (string)	15, 3.14, -123.456E+10
int	Число с необязательным знаком, без дробной части, без показателя степени	32-битное, двоичное, со знаком	1, 58502, -13
float	Такой же, как и для number	IEEE 488 64-битное	.314159265358979E+1
fixed.14.4	Такой же, как number, но не более 14 цифр слева от десятичной точки, и не более 4 цифр справа	64-битное, двоичное, со знаком	12.0044
boolean	“1” или “0”	Бит	0, 1 (1==”true”)
dateTime.iso8601	Дата в формате ISO 8601 с необязательным указанием времени, без указания зоны. Дробные секунды могут быть с точностью до наносекунд	Структура или объект, содержащие год, месяц, час, минуты, секунды, наносекунды	19941105T08:15:00301

Таблица D.1 (продолжение)

Имя	Тип в разборе	Тип в хранении	Примеры
dateTime.iso8601tz	Дата в формате ISO 8601 с необязательным указанием времени и необязательным указанием зоны. Дробные секунды могут быть с точностью до наносекунд	Структура или объект, содержащие год, месяц, час, минуты, секунды, наносекунды, временную зону	19941105T08:15:5+03
date.iso8601	Дата в формате ISO 8601 (без времени)	Структура или объект, содержащие год, месяц, день	19541022
time.iso8601	Время в формате ISO 8601, без даты и без временной зоны	Структура или объект, показывающие день, час, минуту	08:15-05:00
time.iso8601.tz	Время в формате ISO 8601, без даты, но с необязательной временной зоной	Структура или объект, содержащие день, час, минуту, часовую зону, минутную зону	1, 255
i1	Число с необязательным знаком, без дробной части, без степени	8-битное, двоичное	1, 703, -32768
i2	-“-	16-битное, двоичное	1, 255
i4	-“-	32-битное, двоичное	1, 703, -32768
i8	-“-	64-битное, двоичное	
ui1	Число, без знака, без дробной части, без степени	8-битное, двоичное, без знака	
ui2	-“-	32-битное, двоичное, без знака	
ui4	-“-	32-битное, двоичное, без знака	
ui8	-“-	64-битное, двоичное, без знака	
r4	Такой же, как number	IEEE 488 4-байтовое, с плавающей точкой	
r8	-“-	IEEE 488 8-байтовое, с плавающей точкой	
float.IEEE.754.64	-“-	IEEE 754 4-байтовое, с плавающей точкой	
float.IEEE.754.32	-“-	IEEE 754 8-байтовое, с плавающей точкой	
uuid	Шестнадцатеричные цифры, представляющие октеты, дефисы необязательны и должны быть проигнорированы	128-байтовая Unix UUID структура	F04DA480-65B9-11d1-A29F-00AA00C14882

Таблица D.1 (окончание)

Имя	Тип в разборе	Тип в хранении	Примеры
uri	Универсальный идентификатор ресурса (Universal Resource Identifier)	Согласно спецификации консорциума W3C	http://www.ics.uci.edu/pub/ietf/uri/draft-fielding-uri-syntax-00.txt http://www.ics.uci.edu/pub/ietf/uri http://www.ietf.org/html.characters/urn-character.html
bin.hex	Шестнадцатеричные цифры, представляющие октеты	Без конкретного размера	
char	Строка (string)	1 символ Unicode (16 бит)	
string.ansi	Строка, содержащая только ASCII символы <= 0xFF	Unicode или однобайтовая строка	This does not look Greek for me

Все типы времени и даты, для которых указано `iso8601`, в действительности используют ограниченный набор форматов, определенный стандартом ISO 8601. При указании года необходимо предоставить 4 цифры. Порядковые даты не используются. О форматах, где встречаются номера недель: допустимо использование только форматов, отсекающих год и месяц (5.2.3.3 d, e и f).

Приложение E
XML DTD для XML-данных

Это определение типа документа (DTD) выбрано из сообщения консорциума W3C, находящегося по адресу:

http://www.w3.org/TR/1998/NOTE-XML-data

```
<!ENTITY % nodeattrs 'id ID #IMPLIED'>

<!-- href введен согласно XML-LINK, но не требуется, пока нет содержания. -->

<!ENTITY % linkattrs
                    'id ID #IMPLIED
                    href CDATA #IMPLIED'>

<!ENTITY % typelinkattrs
                    'id ID #IMPLIED
                    type CDATA #IMPLIED'>

<!ENTITY % exattrs
                    'name CDATA #IMPLIED
                    content (OPEN | CLOSED) "OPEN" >

<!ENTITY % elementTypeElementsl
                    'genus? correlative? superType*'>

<!ENTITY % elementTypeElements2
                    'description,
                    (min|minExclusive)?,
                    (max | maxinclusive)?,
                    domain*,
                    key*,
                    foreignKey*,
                    (datatype | ( syntax?, objecttype+ ) )?
                    mapsTo?'>

<!ENTITY % elementConstraints
                    'min? max? default?'>

<!ENTITY % elementAttrs
                    'occurs (REQUIRED | OPTIONAL | ONEORMORE | ZEROORMORE) "REQUIRED" '
>

<!ENTITY % rangeAttribute
                    'range CDATA #IMPLIED' >
```

```
<!-- Контейнер верхнего уровня. -->
<!element schema       ((elementType|linkType|
                       extendType|
                       intEntityDcl|extEntityDcl|
                       notationDcl|extDcIs)*)>
<!attlist schema %nodeattrs;>

<!-- Описания типа элемента. -->

<!element elementType  (%elementTypeElements1;,
                       ((element|group)*|empty|any|string|mixed)?,
                       attribute*
                       %elementTypeElements2 )>

<!attlist elementType  %nodeattrs;
                       %exattrs >

<!-- Типы элементов, допускаемые в модели содержания. -->

<!-- Заметим, что эта запись выглядит так коротко -->
<!-- только для модельной группы со всего одним элементом. -->
<!element element      (%elementConstraints;) >

<!-- Этот тип обязательный. -->
<!attlist element      %typelinkattrs;
                       %elementAttrs;
                       presence (FIXED) #IMPLIED >

<!-- Группа в модели содержания: и (and), -->
<!-- последовательностная (sequential) -->
<!-- или альтернативная (disjunctive). -->

<!element group        ((group | element)+)>
<!attlist group        %nodeattrs;
                       %elementattrs;
                       presence (FIXED) #IMPLIED
                       groupOrder (AND|SEQ|OR) 'SEQ'>

<!element any          EMPTY>
<!element empty        EMPTY>
<!element string       EMPTY>

<!-- Смешанное содержание представляет собой просто -->
<!-- плоский, непустой список элементов. -->

<!-- Нет необходимости предоставлять информацию -->
<!-- о <string/> (CDATA): она подразумевается. -->

<!element mixed        (element+)>
<!attlist mixed        %nodeattrs;>

<!element superType    EMPTY>
<!attlist superType    %linkattrs;>
<!element genus        EMPTY>
<!attlist genus        %typelinkattrs;>

<!element description  MIXED>
<!attlist description  %nodeattrs;>
```

```
<!element domain            EMPTY>
<!attlist domain            %typelinkattrs;>

<!element default           MIXED>
<!attlist default           %nodeattrs;>

<!element min               MIXED>
<!attlist min               %nodeattrs; >

<!element max               MIXED>
<!attlist max               %nodeattrs; >

<!element maxInclusive      MIXED>
<!attlist maxinclusive      %nodeattrs; >

<!element minExclusive      MIXED>
<!attlist minExclusive      %nodeattrs; >

<!element key               (keyPart+)>
<!attlist key               %nodeattrs;>

<!element keyPart           EMPTY>
<!attlist keyPart           %linkattrs;>

<!element foreignKey        foreignKeyPart* >
<!attlist foreignKey        %nodeattrs;
                            %rangeAttribute;
                            key CDATA #IMPLIED >

<!element foreignKeyPart    EMPTY>
<!attlist foreignKeyPart    %linkattrs;>

<!-- Поддержка типов данных (datatype). -->

<!element datatype          (elementType?) >
<!attlist datatype          %typelinkattrs;
                            presence (IMPLIED | SPECIFIED | REQUIRED | FIXED) #IMPLIED
>

<!element syntax >
<!attlist syntax            %linkattrs; >

<!element objecttype >
<!attlist objecttype        %linkattrs; >

<!-- Поддержка отображения (mapping). -->

<!element mapsTo            (implies?)>
<!attlist mapsTo            %typelinkattrs;>


<!element implies           (implies?)>
<!attlist implies           %typelinkattrs;>

<!-- Поддержка псевдонима (alias). -->

<!element elementTypeEquivalent EMPTY>
<!attlist elementTypeEquivalent %typelinkattrs; >

<!element correlative       EMPTY>
```

```
<!attlist correlative      %linkattrs;>

<!-- Подтип типа элемента (ElementType), -->
<!-- который явно является отношением. -->

<!element relationType (%elementTypeElements1;,
                        ((element | group)*|empty|any|string | mixed)?,
                        attribute*
                        %elementTypeElements2)>
<!attlist relationType %nodeattrs;
%exattrs; >

<!-- Атрибуты. -->

<!-- Значение по умолчанию должно присутствовать, -->
<!-- если атрибут "presence" определен как "SPECIFIED" -->
<!-- или "FIXED". -->
<!-- Атрибут "presence" по умолчанию определен -->
<!-- как "SPECIFIED", если значения по умолчанию присутствуют, -->
<!-- в противном случае он определен как "IMPLIED". -->

<!element attribute    (%PropertyElements1,
                        %PropertyElements2,
                        %elementConstraints)>
<!attlist attribute    %typelinkattrs;
                        name CDATA #IMPLIED
                        %elementAttrs
                        dt                      CDATA # IMPLIED
                        atttype                 (URIREF|ID|IDREF|IDREFS|ENTITY|ENTI
TIES|
                                                NMTOKEN|NMTOKENS|ENUMERATION|NOTATI
ON|
                                                CDATA) CDATA
                        %rangeAttribute;
                        default CDATA #IMPLIED
                        values MMTOKENS #IMPLIED
                        presence (IMPLIED|SPECIFIED|REQUIRED|FIXED) #IMPLIED >

<!-- Описание обозначений и компонентов. -->
<!-- Примечание: если сделана подобная запись, -->
<!-- только внешние компоненты могут иметь структуры -->
<!-- без необходимости обходить ее. -->
<!-- 'par' равно TRUE для параметрических компонентов. -->
<!-- Идентификаторы 'systemID' and 'publicId' (если они -->
<!-- наличествуют) должны иметь требуемый синтаксис. -->

<!ENTITY % notationattrs   '%nodeattrs
                           systemID CDATA #IMPLIED
                           publicID CDATA #IMPLIED'>

<!ENTITY % entityattrs     '%notationattrs
                           name CDATA #IMPLIED
                           par                  (TRUE | FALSE) "FALSE">

<!-- Описания обозначений (notation). -->
```

```
<!element notationDcl      EMPTY>
<!attlist notationDcl      %notationattrs>
<!element intEntityDcl     PCDATA>
<!attlist intEntityDcl     %entityattrs; >

<!-- Если присутствует обозначение (notation), -->
<!-- компонент рассматривается как бинарный. -->
<ielement extEntityDcl     EMPTY>
<!attlist extEntityDcl     %entityattrs;
                           notation CDATA #IMPLIED>

<!-- Внешний компонент с описаниями, -->
<!-- которые следует подставить. -->
<!element extDcIs          EMPTY>
<!attlist extDcIs          %entityattrs;>
```

Приложение F
Свойства каскадных таблиц стилей CSS1

Мы намеренно поместили свойства каскадных таблиц стилей CSS1 и CSS2 в разные разделы, поскольку свойства CSS1 практически полностью реализованы в двух основных браузерах, в то время как свойства CSS2 (за исключением некоторых свойств позиционирования) – нет. Если вы находите в данном приложении некоторое свойство, вы вполне можете быть уверены, что его использование приведет к соответствующим результатам в обоих основных браузерах.

Мы отметили свойства, не особенно удачно реализованные в браузерах Communicator 4 или IE4, но поскольку реализация каскадных таблиц стилей постоянно обновляется, наши замечания могут оказаться не соВсем верными.

Далее приведен список свойств из «Рекомендации по каскадным таблицам стилей CSS1». Мы придерживались композиционного построения, принятого в рекомендации, поэтому описания свойств в данном приложении разбиты на следующие разделы:

- свойства шрифтов,
- свойства цвета и фона,
- свойства текста,
- свойства рамок
- свойства классификации.

Более подробно с проектом можно ознакомиться по адресу:

http://www.w3.org/TR/REC-CSS1

По мере освоения того или иного набора свойств мы будем давать ссылку на соответствующую часть спецификации. Более подробно о каждом свойстве можно узнать из спецификации.

Свойства шрифтов

Эти свойства (см. табл. F.1) описаны в разделе 5.2 спецификации каскадных таблиц стилей CSS1.

Таблица F.1

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
font-family (семейство шрифтов)	font-family: [[<family-name> | <generic-name>],]* [<family-name> | <generic-name>]	Используйте любое название семейства шрифтов. Значения <generic-name>: serif sans-serif *cursive* ***fantasy*** `monospace`	Начальное значение устанавливается браузером	Всем элементам	Да
font-style (стиль шрифта)	font-style: <значение>	normal | *italic* | ***oblique*** (нормальный, *курсив*, ***наклонный***)	normal	Всем элементам	Да
font-variant (разновидность шрифта)	font-variant: <значение>	normal | SMALLCAPS (нормальный, КАПИТЕЛЬ)	normal	Всем элементам	Да
font-weight (насыщенность шрифта)	font-weight: <значение>	normal | **bold** | **bolder** | lighter (нормальный, полужирный, жирный, светлый) |100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900	normal	Всем элементам	Да
'font-size' (размер шрифта)	font-size: <значение> значение= <absolute-size> | <relative-size> | <length> | <percentage> (абсолютный размер, относительный размер, длина, процентное отношение)	<absolute-size> xx-small |x-small |small | medium | large | x-large | xx-large (small – маленький,medium – средний, large – крупный, x– очень, xx– самый) <relative-size>:- larger | smaller (больше, меньше) <length> | <percentage>:- по отношению к родительскому элементу	medium	Всем элементам	Да
font (шрифт)	font: <значение>	[<font-style> || <font-variant> || <font-weight>]? <font-size> [/<line-height>] <font-family> (line-height – высота строки)	Не определено	Всем элементам	Да

Свойства цвета и фона

Эти свойства (см. табл. F.2) описаны в разделе 5.3 спецификации каскадных таблиц стилей CSS1.

Таблица F.2

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
color (цвет)	color: <значение>	keyword | numerical RGB specification (ключевое слово или числовая RGB спецификация)			
background-color (цвет фона)	background-color: <значение>	<color> | transparent (цвет, прозрачный)	transparent	Всем элементам	Нет
background-image (рисунок фона)	background-image: <значение>	<url> | none (<url>, или ничего)	none	Элементам, блокам	Нет
background-repeat (размножение фона). Неудачно реализовано в браузере Communicator 4	background-repeat: <значение>	repeat | repeat-x | repeat-y | no-repeat (размножение, ..по x, ..по y, без размножения)	repeat	Пунктам списка	Нет
background-attachment (закрепление фона) не поддерживается в браузере Communicator 4	background-attachment: <значение>	scroll |fixed (прокручиваемый, неизменный)	scroll	Пунктам списка	Нет
background-position (расположение фона). Неудачно реализовано в браузере Communicator 4	background-position: <значение>	[<length> | <percentage>] {1,2} | [top | center | bottom] || [left | center | right] (длина, процентное отношение, верх, центр, низ, слева, центр, справа)	0%, 0%	Пунктам списка	Нет
background (фон)	background: <значение>	<background-color> || < ackground-image> || <background-repeat> || background-attachment> || <background-position>	Не определено	Пунктам списка	Нет

Свойства текста

Эти свойства (см. табл. F.3) описаны в разделе 5.4 спецификации каскадных таблиц стилей CSS1.

Таблица F.3

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
letter-spacing (интервал) Нет в Communicator 4	letter-spacing: <значение>	normal | <length> (нормальный, длина)	normal	Всем элементам	Да
text-decoration (эффекты)	text-decoration: <значение>	none | [underline || overline || linethrough || blink} (Нет, под-черкнутый, линия сверху, зачеркнутый, мерцающий)	none	Всем элементам	Да
vertical-align (выравнивание по вертикали). Реализовано частично и неудачно	vertical-align: <значение>	baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage> (по опорной линии, подстрочный, надстрочный, по верху, по верху текста, по середине, по низу, по низу текста, процент)	baseline	Элементам в строке inline elements)	Нет
text-transform (преобразование текста)	text-transform: <значение>	none | Capitalize | UPPER_CASE | lower_case (Нет, с заглавной буквы, все заглавными, все строчными)	none	Всем элементам	Да
text-align (выравнивание текста)	text-align: <значение>	left | right | center | justify (по левому краю, по правому, по центру, по ширине). Значение justify не поддерживается в браузере Communicator 4	left	Элементам блока	Да
text-indent (отступ текста)	text-indent: <значение>	<length> | <percentage> (длина, процент)	0	Элементам блока	Да
line-height (высота строки)	line-height: <значение>	normal | <number> | <length> | <percentage> (нормальная, число, длина, процент) <number>:- line-height= font-size x num <percentage>: по отношению к размеру шрифта	normal	Всем элементам	Да

Свойства рамок

Эти свойства (см. табл. F.4) описаны в разделе 5.5 спецификации каскадных таблиц стилей CSS1.

Таблица F.4

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
margin-top (верхнее поле)	margin-top: <значение>	<length> | <percentage> | auto (длина, процент, авто) <percentage>: от ширины родительских элементов. Отрицательные значения допустимы.	0	Всем элементам	Нет
margin-right (правое поле)	margin-right: <значение>	-:то же:-	0	Всем элементам	Нет
margin-bottom (нижнее поле)	margin-bottom: <значение>	-:то же:-	0	Всем элементам	Нет
margin-left (левое поле)	margin- left: <значение>	-:то же:-	0	Всем элементам	Нет
margin (поле)	margin: <значение>	[<length> | <percentage> | auto] {1,4} (длина, процент, авто). Если даны 4 величины, они применяются по порядку к верхнему, правому, нижнему, левому полям. Одна величина – ко всем 4 полям. Если даны 2 или 3, то отсутствующие величины устанавливаются равными величинам для противоположных полей. <percentage>: от ширины родительских элементов. Отрицательные значения допустимы	не определено	Всем элементам	Нет

Таблица F.4 (продолжение)

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
padding-top (верхнее заполнение)	padding-top: <значение>	[<length> | <percentage> (длина, процент) <percentage>: от ширины родительских элементов. Отрицательные значения недопустимы	0	Всем элементам	Нет
padding-right правый край заполнения)	padding-right: <значение>	-:то же:-	0	Всем элементам	Нет
padding-bottom (нижнее заполнения)	padding-bottom: <значение>	-:то же:-	0	Всем элементам	Нет
padding-left (левый край заполнения)	padding-left: <значение>	-:то же:-	0	Всем элементам	Нет
padding (заполнение)	padding: <значение>	[<length> | <percentage> | auto] {1,4} (длина, процент, авто) Если даны 4 величины, они применяются по порядку к верхнему, правому, нижнему, левому краям. Одна величина – ко всем 4 краям. Если даны 2 или 3, то отсутствующие величины устанавливаются равными величинам для противоположных краев. <percentage>: от ширины элементов. Отрицательные значения недопустимы	0	Всем элементам	Нет
border-top-width (ширина верхней линии обрамления)	border-top-width: <значение>	thin | medium | thick | <length> (тонкая, средняя, толстая, длина)	medium	Всем элементам	Нет

Таблица F.4 (продолжение)

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
border-right-width (ширина правой линии обрамления)	border-right-width: <значение>	thin | medium | thick | <length> (тонкая, средняя, толстая, длина)	medium	Всем элементам	Нет
border-bottom-width (ширина нижней линии обрамления)	border-bottom-width: <значение>	thin | medium | thick | <length> (тонкая, средняя, толстая, длина)	medium	Всем элементам	Нет
border-left-width (ширина левой линии обрамления)	border-left-width: <значение>	thin | medium | thick | <length> (тонкая, средняя, толстая, длина)	medium	Всем элементам	Нет
border-width (ширина линии обрамления)	border-width: <значение>	thin | medium | thick | <length> (1,4} (тонкая, средняя, толстая, длина). Если даны 4 величины, они применяются по порядку к верхней, правой, нижней, левой линиям обрамления. Одна величина – ко Всем 4 линиям. Если даны 2 или 3, то отсутст-вующие величины устанавливаются равными величинам для противоположных линий	Не определено	Всем элементам	Нет
border-color (цвет обрамления)	border-color: <значение>	<color>{1,4} (см. дополнение Е). Если даны 4 величины, они применяются по порядку к верхней, правой, нижней, левой линиям обрамления (в таком порядке). Одна величина – ко всем 4 линиям. Если даны 2 или 3, то отсутст-вующие величины устанавливаются равными величинам для противоположных линий	Свойство цвета элемента	Всем элементам	Нет

Таблица F.4 (окончание)

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
border-style (стиль обрамления)	border-style: <значение>	[none | dotted | dashed | solid | double | groove | ridge | inset | outset] {1,4} (Нет, пунктир, штрих, сплошная, двойная, волнистая, загзагообразная, внутренний боковик, внешний боковик)	none	Всем элементам	Нет
border-top (обрамление сверху)	border-top: <значение>	<border-top-width> || <border-style> || <color>	Не определено	Всем элементам	Нет
border-right (обрамление справа)	border-right: <значение>	<border-right-width> || <border-style> || <color>	Не определено	Всем элементам	Нет
border-bottom (обрамление снизу)	border-bottom: <значение>	<border-bottom-width> || <border-style> || <color>	Не определено	Всем элементам	Нет
border-left (обрамление слева)	border-left: <значение>	<border-left-width> || <border-style> || <color>	Не определено	Всем элементам	Нет
border (обрамление)	border: <значение>	<border-width> || <border-style> || <color>	Не определено	Всем элементам	Нет
width (ширина)	width: <значение>	<length> | <percentage> | auto (длина, процент, авто) <percentage>: от ширины родительских элементов. Замещаемый элемент – такой как IMG или OBJECT в HTML	auto	Элементам блоков и замещаемым элементам	Нет
height (высота)	height: <значение>	<length> | auto (длина, авто)	auto	элементам блоков и замещаемым элементам	Нет
float (свободно перемещаемый)	float: <значение>	left | right | none (слева, справа, Нет) Внимание: свойство float удаляет встроенные (inline) элементы из строки	none	Всем элементам	Нет
clear (пустой)	clear: <значение>	block | inline | list-item | none (блок, в строке, пункт списка, Нет)	none	Всем элементам	Нет

Свойства классификации

Эти свойства (см. табл. F.5) описаны в разделе 5.6 спецификации каскадных таблиц стилей CSS1.

Таблица F.5

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
display (отображение)	display: <значение>	block | inline | list-item | none (блок, в строчке, пункты списка, нет)	Зависит от браузера. Если тэг неизвестен, как в случае с XML, то обычно по умолчанию – inline, хотя для CSS1 значение по умолчанию – block	Всем элементам	Нет
white-space (пробельные литеры)	white-space: <значение>	normal | pre | nowrap (нормальный, сохраняется, не переносится)	normal	Элементам, блокам	Нет
list-style-type (тип стиля списка)	list-style-type: <значение>	disc | circle | square | decimal | lower-roman | upper-roman | lower-alpha | upper-alpha | none (диск, круг, квадрат, десятичный, строчная римская цифра, заглавная римская цифра, строчная латинская буква, заглавная латинская буква, нет)	disc	Пунктам списка	Да
list-style-image (изображение стиля списка)	list-style-image: <значение>	<url> | none (URL адрес, нет)	none	Пунктам списка	Да
list-style-position (расположение стиля списка)	list-style-position: <значение>	inside | outside (внутри, вне)	outside	Пунктам списка	Да
list-style (стиль списка)	list-style: <значение>	<list-style-type? || <list-style-position> || <url>	Не определено	Пунктам списка	Да

Приложение G
Свойства каскадных таблиц стилей CSS2

Каскадные таблицы стилей второго уровня (CSS2) включают в себя все свойства каскадных таблиц стилей первого уровня (CSS1), хотя во многих случаях добавлены новые значения. Большинство новых свойств CSS2 не реализовано в современных браузерах, за исключением практически всех свойств позиционирования. Однако свойства абсолютного позиционирования не реализованы в браузере Communicator 4.

Данное дополнение придерживается такой же системы разбиения на разделы, как это принято в официальной спецификации:

- модель рамок (Box Model);
- модель визуального представления (Visual Rendering Model);
- детали модели визуального представления (Visual Rendering Model Details);
- визуальные эффекты (Visual Effects);
- сгенерированное содержание и автоматическая нумерация (Generated Content and Automatic Numbering);
- носители; разбитые на страницы (Paged Media);
- цвет и фон (Colors and Backgrounds);
- свойства шрифтов (Font Properties);
- свойства текста (Text Properties);
- таблицы списков (Lists Tables);
- интерфейс пользователя (User Interface);
- звуковые таблицы стилей (Aural Style Sheets).

Там, где свойства CSS2 не отличаются от свойств CSS1, просто дается ссылка на дополнение F, приведенное в этой книге.

Более подробную информацию по каждому свойству можно найти в соответствующих разделах спецификации; ссылки на эти разделы даны в тексте. С официальной спецификацией можно ознакомиться по адресу:

http://www.w3.org/TR/PR-CSS2

Как и положено, чтобы выяснить, работает ли данное свойство или нет, его следует просто применить на практике.

Порядок каскадирования – это единственная область, где CSS2 отличаются от CSS1. В CSS1 свойства, описанные в таблице авторских стилей с применением ключевого слова `!important`, предшествовали аналогично описанным свойствам в пользовательской таблице стилей. В CSS2 этот порядок был изменен на обратный (что, безусловно, вернее).

Модель рамок

В каскадных таблицах стилей второго уровня (CSS2) свойства рамок по сравнению с CSS1 не изменились; детальная информация о названиях свойств и их значениях приведена в дополнении F.

Модель визуального представления

Модель визуального представления (см. табл. G.1) – это новая категория свойств в каскадных таблицах стилей второго уровня. Она сгруппирована из новых и уже реализованных в CSS1 свойств.

Два существующих в CSS1 и не изменившихся свойства, `float` (плавающий) и `clear` (пустой), полностью описаны в дополнении F.

В CSS2, по сравнению с CSS1, к свойству `display` (отображение) добавлены ниже перечисленные значения. В других отношениях это свойство не изменилось.

`run-in` (вставка) | `compact` (компактный) | `marker` (маркер) | `table` (таблица) | `inline-table` (таблица в строке) | `table-row-group` (группа строк таблицы) | `table-column-group` (группа столбцов таблицы) | `table-header-group` (группа верхних колонтитулов таблицы) | `table-footer-group` (группа нижних колонтитулов таблицы) | `table-row` (строка таблицы) | `table-cell` (ячейка таблицы) | `table-caption` (заголовок таблицы).

Два новых свойства, `position` (позиционирование) и `z-index` (z-индекс), описаны ниже. Эти свойства по-разному реализованы в браузерах IE4 и Communicator 4. (Браузер Netscape не поддерживает значение `absolute` при позиционировании. Вместо этого следует использовать свойство `<LAYER>` (слой).) Оба коллектива разработчиков обещают обеспечить полную поддержку свойств в 5-х версиях браузеров.

Полностью модель визуального представления раскрыта в разделе 9 спецификации каскадных таблиц стилей второго уровня.

Детали модели визуального представления

Детали модели визуального представления (см. табл. G.2) – это новый раздел в каскадных таблицах стилей второго уровня (CSS2). К свойствам `width` (ширина), `height` (высота), `line-height` (высота строки), не изменившимися по сравнению с CSS1 (см. дополнение F), добавлены четыре новых свойства: `min-height`, `max-height`, `min-width` и `max-width`. Описания этих свойств даны ниже.

Данная информация полностью раскрыта в разделе 10 спецификации CSS2.

Таблица G.1

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
position (расположение)	position: <значение>	static | relative | absolute | fixed (неподвижный, относительный, абсолютный, закрепленный)	static	Всем элементам	Нет
box offsets (смещение рамки)	box offsets: [top | left | bottom | right] (вверху, слева, снизу, справа)	<length> | <percentage> | auto (длина, процент, авто)	auto	Всем элементам	Нет
z-index (z-индекс)	auto | <integer> (авто, целый)	<length>: постоянное смещение от опорного края. <percentage>: смещение в процентах от ширины содержащего блока (для значений left и right) или от высоты (для значений top и bottom)	auto	Элементам, отображаемым рамками, позиционированными абсолютно или относительно	Нет

Таблица G.2

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
min-width (минимальная ширина)	min-width: <значение>	<length> | <percentage> (длина, процент)	0	Всем элементам	Нет
max-width (максимальная ширина)	max-width: <значение>	<length> | <percentage> (длина, процент)	100%	Всем элементам	Нет
min-height (минимальная высота)	min-height: <значение>	<length> | <percentage> (длина, процент)	0	Всем элементам	Нет
max-height (максимальная высота)	max-height: <значение>	<length> | <percentage> (длина, процент)	100%	Всем элементам	Нет

Визуальные эффекты

Визуальные эффекты (см. табл. G.3) – это новая категория свойств в каскадных таблицах стилей второго уровня (CSS2). Полностью информация раскрыта в разделе 11 спецификации CSS2.

Таблица G.3

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
overflow (переполнение)	См. спецификацию	visible | hidden | scroll | auto (видимый, скрытый, прокручиваемый, авто)	visible	Элементам уровня блока и замещаемым элементам	Нет
clip (усечение)	См. спецификацию	<shape> | auto (форма, авто)	auto	Элементам уровня блока и замещаемым элементам	Нет
visibility (видимость)	visibility: <значение>	visible | hidden | collapse | inherit (видимый, скрытый, сжатый, унаследованный)	inherit	Всем элементам	Нет

Сгенерированное содержание и автоматическая нумерация

Сгенерированное содержание и автоматическая нумерация (см. табл. G.4) – это новая категория свойств в каскадных таблицах стилей второго уровня (CSS2). Полная информация дана в разделе 12 спецификации CSS2. В CSS2 содержание можно создавать несколькими способами:

- используя свойство `content` (содержание) совместно с псевдоэлементами `:before` (до) и `:after` (после);
- совместно со звуковыми свойствами `cue-before` (воспроизведение до) и `cue-after` (воспроизведение после);
- при помощи элементов со значениями `list-item` (пункт списка) для свойства `display` (отображение).

Стиль и местоположение сгенерированного содержания даются псевдоэлементами `:before` и `:after`. Эти псевдоэлементы используются совместно со свойством `content`, которое задает, какой именно текст вставляется. Псевдоэлементы `:before` и `:after` задают содержание до и после содержания дерева документа элементов. Дополнительная информация по данному вопросу тоже дана в разделе 12 спецификации CSS2.

Таблица G.4

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
content (содержание)	См. спецификацию	<string> | <uri> | <counter> | attr(X) | open-quote | close-quote | no-open-quote | no-close-quote]+ (строка, uri, счетчик, значение атрибута X, открывающие кавычки, закрывающие кавычки, нет открывающих кавычек, нет закрывающих кавычек)	empty string (пустая строка)	Псевдоэлементам :before (до) и :after (после)	Нет
quotes (кавычки)	См. в спецификации, как следует указывать свойство quotes и как вставлять кавычки при использовании свойства content	[<string>.,<string>.]+ | none | inherit (строка, строка, нет, унаследованный)	Зависит от агента пользователя	Всем элементам	Да

Носители, разбитые на страницы

Все свойства разбитых на страницы (см. табл. G.5) носителей являются новыми в каскадных таблицах стилей 2 (CSS2). Полностью они описаны в разделе 13 спецификации CSS2.

Таблица G.5

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
size (размер)	size: <значение>	<length>{1,2} | auto | portrait | landscape (длина, авто, книжный, альбомный)	auto	Содержанию страницы	Не определено
marks (crop marks) (метки, метки обреза)	marks: <значение>	[crop || cross] | none (отрезание, совмещение, нет)	none	Содержанию страницы	Не определено
page (страница) (для использования страниц, имеющих имена)	page: <идентификатор> <значение>	[left | right]? | auto	auto	Элементам уровня блока	Да
page-break-before (разрыв страницы до)	page-break-before: <значение>	auto | always | avoid | left | right | inherit (авто, всегда, избегать, как левая, как правая, унаследованное)	auto	Элементам уровня блока	Да
page-break-after (разрыв страницы после)	page-break-after: <значение>	auto | always | avoid | left | right | inherit (авто, всегда, избегать, как левая, как правая, унаследовано)	auto	Элементам уровня блока	Да
page-break-inside (разрыв страницы внутри)	page-break-inside: <значение>	avoid | auto (избегать, авто)	auto	Элементам уровня блока	Да
orphans (висячие строки на новой странице)	orphans: <значение>		2	Элементам уровня блока	Да
widows (висячие строки на предыдущей странице)	widows: <значение>		2	Элементам уровня блока	Да

Цвет и фон

Изменений в этой категории свойств в каскадных таблицах второго уровня (CSS2) по сравнению с CSS1 не произошло. Чтобы ознакомиться с данными свойствами в спецификации каскадных таблиц стилей CSS2, см. раздел 14, в данной книге эти свойства приведены в дополнении F.

Свойства шрифтов

Все свойства шрифтов (см. табл. G.6), реализованные в каскадных таблицах стилей первого уровня (CSS1), остались без изменения в CSS2, они описаны в дополнении F. Однако в CSS2 добавлены два новых свойства шрифтов. Полностью свойства шрифтов приведены в разделе 15 спецификации CSS2.

Таблица G.6

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
font-stretch (растяжение шрифта) не поддерживается	font-stretch: <значение>	normal | wider | narrower | ultra-condensed | extra-condensed | condensed | semi-condensed | semi-expanded | expanded | extra-expanded | ultra-expanded (нормальный, шире, уже, крайне уплотненный, очень уплотненный, уплотненный, полууплотненный, полуразреженный, разреженный, очень разреженный, крайне разреженный)	normal	Всем элементам	Да
font-size-adjust (настройка размера шрифта)	font-size-adjust: <значение>	<number> | none (число, нет)	none	Всем элементам	Да

Свойства текста

Все свойства текста (см. табл. G.7), реализованные в каскадных таблицах стилей первого уровня (CSS1), остались без изменения в CSS2; они также описаны в дополнении F. Однако в CSS2 добавлено одно новое свойство. Полностью свойства текста приведены в разделе 16 спецификации CSS2.

Таблица G.7

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
text-shadow (тень текста)	text-shadow: <значение>	none | [<color> || <length> <length> <length>? ,]* [<color> || <length> <length> <length>?] | inherit (none- нет, color- цвет, length- длина, inherit- унаследованный)	none	Всем элементам	Нет

Списки

Все свойства списков (см. табл. G.8), реализованные в каскадных таблицах стилей первого уровня (CSS1) и описанные в дополнении F, остались без изменения в CSS2, за исключением свойства `list-style-type` (тип стиля списка). Полностью свойства списков приведены в разделе 17 спецификации CSS2.

Таблица G.8

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
list-style-type (тип стиля списка)	list-style-type: <значение>	leading-zero | western-decimal | lower-latin | upper-latin | hebrew | armenian | georgian | cjk-ideographic | hiragana | katakana | hiragana-iroha | katakana-iroha (ведущий ноль, десятичный, строчные ascii буквы, заглавные ascii буквы, иврит, армянский, грузинский, идеографический, хирагана, катакана, хирагана-ироха, катакана-ироха)	disc	Элементам, у которых свойство display установлено равным list-item	Да

Таблицы

Все свойства `table` (таблица) – см. табл. G.9 – в каскадных таблицах стилей второго уровня (CSS2) являются новыми. Их описания даны в разделе 18 спецификации CSS2.

Таблица G.9

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
column-span (охват столбцов)	column-span: <целое значение>		1	Элементам table-cell, table-column, table-column-group (ячейка таблицы, столбец таблицы, группа столбцов таблицы)	Нет
row-span (охват строк)	row-span: <целое значение>		1	Элементам table-cell (ячейка таблицы)	Нет
table-layout (компоновка таблицы)	table-layout: <значение>	fixed | auto (неизменный, авто)	auto	Элементам table, inline-table (таблица, таблица в строке)	Нет
empty-cells (пустые ячейки)	empty-cells: <значение>	borders | no-borders (обрамления, нет обрамлений)	borders	Элементам table-cell (ячейка таблицы)	Да
speak-header (произносить заголовок)	speak-header: <значение>	once | always (один раз, всегда)	once	Элементам, имеющим информацию в заголовках	Да

Интерфейс пользователя

Все свойства интерфейса пользователя в каскадных таблицах стилей второго уровня являются новыми (см. табл. G.10). Их описания даны в разделе 19 спецификации CSS2.

Таблица G.10

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
cursor: <uri> <value>	[[<uri>,]* [auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize| text | wait | help]] (uri, авто, перекрестье, по-умолчанию, указатель, перемещение, (resize- изменение размера, e- вправо, ne- вверх и вправо, nw- вверх и влево, n- вверх, se- вниз и вправо, sw- вниз и влево, s- вниз, w- вниз), текст, ожидание, вопрос)	auto	auto	Всем элементам	Да
outline: [<outline-color> || <outline-style> || <outline-width>]	[<outline-color> || <outline-style> || <outline-width>]	См. значения каждого свойства	См. значения каждого свойства	Всем элементам	Нет
outline-width: <value>	border-width (ширина линии обрамления)	medium (средний)	medium (средний)	Всем элементам	Не определено
outline- style: <value>	border- style (стиль_ обрамления)	none (нет)	none (нет)	Всем элементам	Не определено
outline-color: <value>	border- color, invert (цвет обрамления, инверсный)	invert	invert	Всем элементам	Не определено

Звуковые таблицы стилей

Звуковые таблицы стилей (см. табл. G.11) – это совершенно новое понятие, введенное в каскадных таблицах стилей второго уровня (CSS2). Более подробная информация находится в разделе 20 спецификации CSS2.

Таблица G.11

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
volume (громкость)	volume: <value>	<number> | <percentage> | silent | x-soft | soft | medium | loud | x-loud (число, процент, нет звука, громкость 0, громкость 25, громкость 50, г ромкость 75, громкость 100)	medium	Всем элементам	Да
speak (произно-сить)	speak: <value>	normal | none | spell-out (нормально, нет, по буквам)	normal	Всем элементам	Да
pause-before (пауза до)	pause-before: <value>	<time> | <percentage> (длительность, процент)	Зависит от агента пользователя	Всем элементам	Нет
pause-after (пауза после)	pause-after: <value>	<time> | <percentage> (длительность, процент)	Зависит от агента пользователя	Всем элементам	Нет
pause (shorthand) (пауза, краткая запись предыдущих двух свойств)	pause: <value>	[[<time> | <percentage>]{1,2}]	Зависит от агента пользователя		
cue-before (воспроиз-ведение до)	cue-before: <value>	<uri> | none (uri, нет)	none	Всем элементам	Нет
cue-after (воспроиз-ведение после)	cue-after: <value>	<uri> | none (uri, нет)	none	Всем элементам	Нет
cue (shorthand) (воспроиз-ведение, краткая запись предыдущих двух свойств)	cue: <value>	[<cue-before> || <cue-after>]	Отсутствует при подобной краткой записи	Всем элементам	Нет

Таблица G.11. (продолжение)

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
play-during (воспроизводить во время)	play-during: <uri> <value>	<uri> mix? repeat? | auto | none | inherit (uri_смешать, повторять, авто, нет, унаследованный)	auto	Всем элементам	Нет
azimuth (азимут)	azimuth: <value>	<angle> | [[left-side | far-left | left | center-left | center | center-right | right | far-right | right-side] || behind] | leftwards | rightwards (угол, левая сторона, крайний левый, слева, левее центра, по центру, правее центра, справа, крайний правый, правая сторона, позади, переместить левее, переместить правее)	center	Всем элементам	Да
elevation (возвышение)	elevation: <value>	<angle> | below | level | above | higher | lower (угол, снизу, на уровне, сверху, выше, ниже)	level	Всем элементам	Да
speech-rate (скорость воспроизведения)	speech-rate: <value>	<number> | x-slow | slow | medium | fast | x-fast | faster | slower (число, очень медленно, медленно, средне, быстро, очень быстро, быстрее, медленнее)	medium	Всем элементам	Да
voice-family (тип голоса) (тип речи, пол говорящего)	voice-family: <value> <value>	[[<specific-voice> | <generic-voice>],]* [<specific-voice> | <generic-voice>]	Зависит от агента пользователя	Всем элементам	Да
pitch (тон)	pitch: <value>	<frequency> | x-low | low | medium | high | x-high (частота, очень низкий, низкий, средний, высокий, очень высокий)	medium	Всем элементам	Да
pitch-range (диапазон тона)	pitch-range: <value>	<number> (число)	50	Всем элементам	Да

Таблица G.11 (окончание)

Название свойства	Синтаксис свойства	Возможные значения	Начальное значение	Применяется к	Наследуется
stress (ударения)	stress: <value>	<number> (число)	50	Всем элементам	Да
richness (богатство голоса)	richness: <value>	<number> (число)	50	Всем элементам	Да
speak-punctuation (произношение знаков пунктуации)	speak-punctuation: <value>	code | none (закодированный, нет)	none	Всем элементам	Да
speak-numeral (произношение числительных)	speak-numeral: <value>	digits | continuous (цифры по отдельности, непрерывно)	continuous	Всем элементам	Да

Приложение H
Поддержка читателей и список опечаток

В книгах по программированию самое большое негодование вызывают не выявленные вовремя ошибки. Код, на набор которого потрачен не один час, вдруг отказывается работать. Вы проверяете его сотни раз, упорно ищете промашку, и наконец обнаруживаете, что имя какой-нибудь переменной написано в книге неверно. Кошмар! Безусловно, вы можете ругать авторов за то, что они плохо проверили код; редакторов за то, что они не обеспечили должного качества; корректоров за то, что их орлиный глаз не выловил всех проколов, но все это делу не поможет, – ошибка уже произошла.

Мы сделали все возможное, чтобы ни один подобный дефект не прокрался в наш сборник, однако дать абсолютную гарантию, что их вообще нет, все-таки нельзя. Чем издатели действительно могут помочь, это предложить наладить обратную связь и немедленную помощь со стороны специалистов, работавших над книгой. Мы обязательно постараемся, чтобы в будущих изданиях все выявленные ошибки были устранены.

Теперь самое время шаг за шагом ознакомить читателей с процессом передачи на наш Web-сайт обнаруженных опечаток. Для этого соответствующий материал разбит на следующие разделы:

- членство в клубе разработчиков Wrox;
- поиск на Web-сайте списка найденных опечаток;
- внесение обнаруженных вами опечаток в существующий список;
- что происходит с вашими замечаниями после того, как вы их послали (почему они не появляются в списке мгновенно).

Далее вы узнаете, как можно получить техническую поддержку по электронной почте. Для этого необходимо знать:

- что должно содержаться в вашем электронном послании;
- что будет происходить с вашим электронным письмом после того, как мы его получим.

Единственное, что вам придется сделать, чтобы иметь возможность просматривать информацию об ошибках и опечатках, это зарегистрироваться в качестве члена клуба разработчиков Wrox. Процедура легкая и быстрая, в дальнейшем официальная причастность к разработке того или иного вопроса сэкономит вам немало

времени. Если вы уже являетесь членом клуба, просто обновите свое членство и включайтесь в работу и над этой книгой.

Членство в клубе разработчиков издательства Wrox

Чтобы *бесплатно* стать членом клуба разработчиков издательства Wrox, щелкните в навигационной панели нашего Web-сайта:

www.wrox.com

по ссылке Membership (членство), как это показано на рис. H.1.

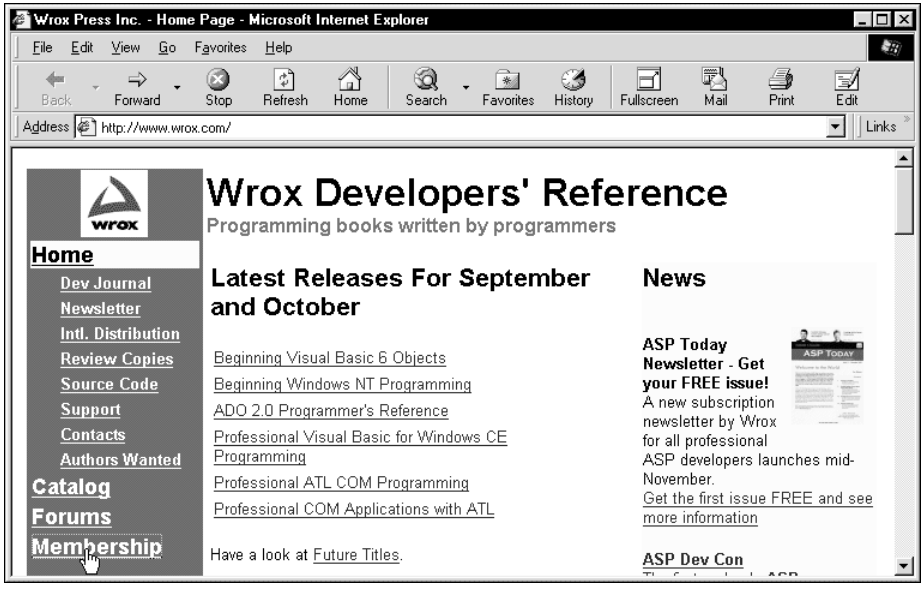


Рис. H.1. Домашняя страница Web-сайта издательства Wrox

Затем щелкните по ссылке **New User** (Новый пользователь). В результате вы окажетесь на странице, содержащей пустой бланк. Заполните его и пошлите введенные вами данные, щелкнув по кнопке **Submit** (Послать), расположенной внизу экрана. Быстрее, чем вы успеете произнести: «В издательстве Wrox выходят в свет лучшие книги по программированию!» – перед вами появится следующая страница (рис. H.2).

Поиск на Web-сайте списка опечаток

Перед тем как послать вопрос, проверьте, нет ли уже ответа на него на Web-сайте издательства по адресу:

http:\\www.wrox.com

Для каждой выпущенной нашим издательством книги предусмотрена отдельная страница со списком обнаруженных опечаток. Перейти к информации по искомой книге вы можете, щелкнув по ссылке **Support** (Поддержка), находящейся слева на навигационной панели.

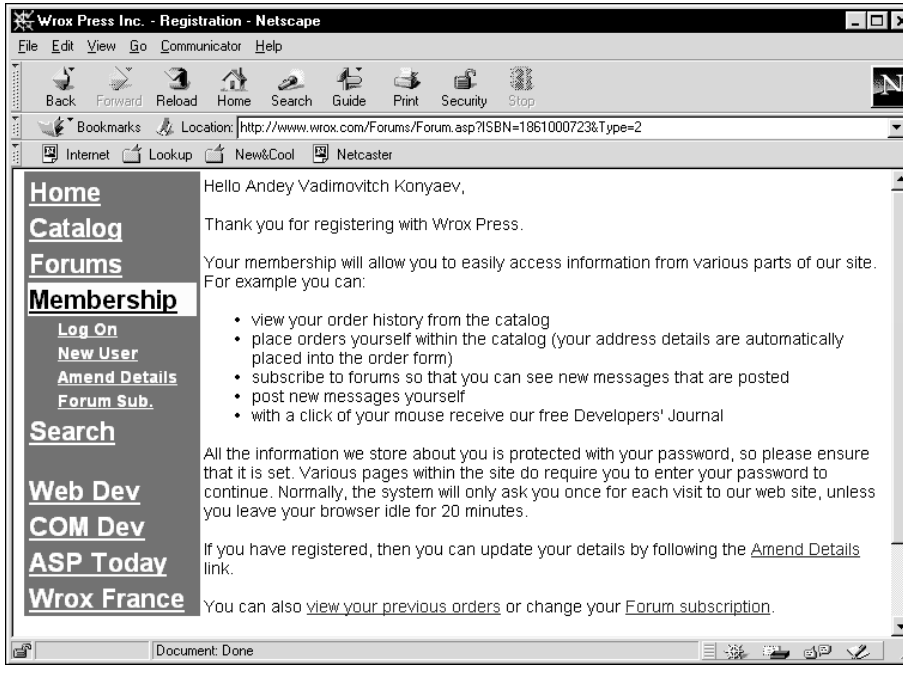


Рис. H.2. Регистрация нового члена клуба разработчиков издательства Wrox

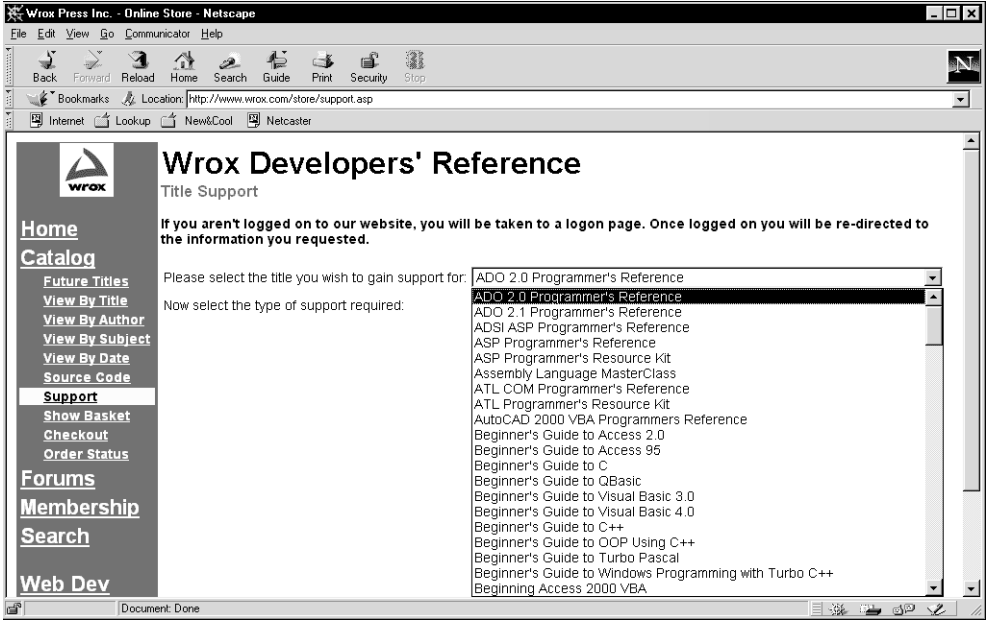


Рис. H.3. Выбор списка опечаток для книг издательства Wrox

Отсюда (рис. Н.3) можно перейти на страницу с опечатками, обнаруженными в любой нашей книге. Выберите в раскрывающемся меню нужную публикацию и щелкните по названию.

Затем щелкните по кнопке **Enter Book Errata** (Перейти к странице с опечатками). Это приведет вас к списку опечаток в интересующей вас книге (рис. Н.4). Выберите критерии, согласно которым вы хотите просмотреть опечатки, и щелкните по кнопке **Apply criteria** (Применить критерии). В результате перед вами появятся ссылки на соответствующие опечатки. Для начала следует заняться поиском по номерам страниц. Если вы уже работали со списком, то поиск можно ограничить по дате. Чтобы обеспечить вас самой свежей информацией о найденных ошибках и опечатках, мы обновляем эти страницы ежедневно.

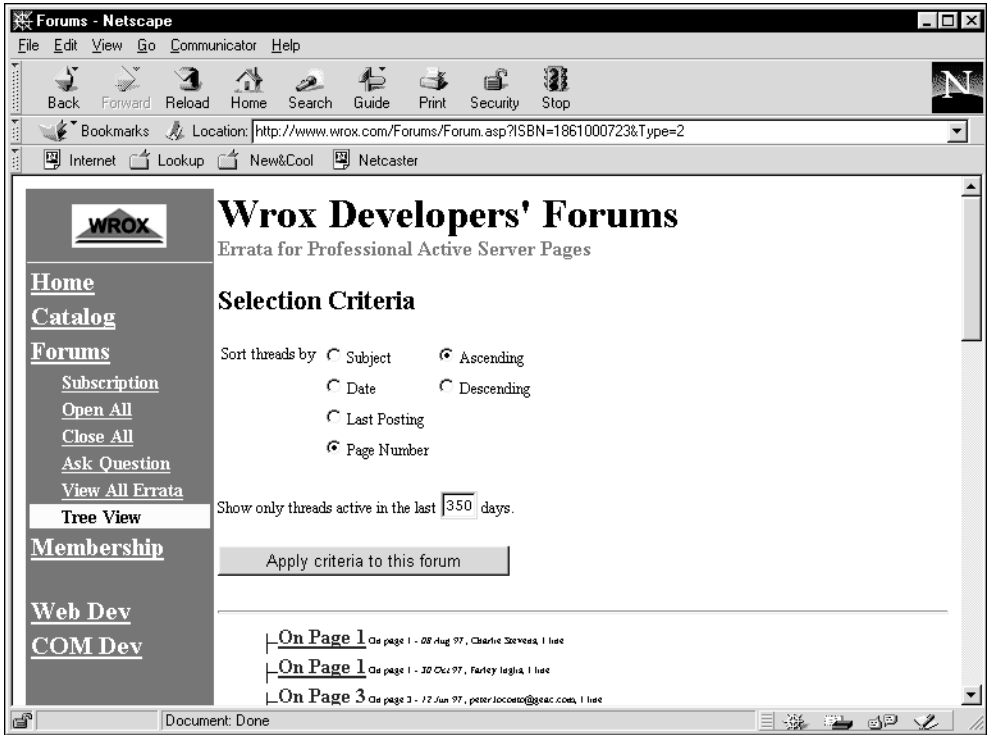


Рис. Н.4. Критерии сортировки при просмотре списка опечаток

Добавление опечаток в существующий список

Вполне вероятно, что выявленная вами ошибка еще не успела войти в наш перечень. В таком случае вы можете сделать это сами. Дефектом может оказаться и обыкновенная орфографическая ошибка, и неверно работающий фрагмент кода из книги. Бывает, что читатели иной раз испытывают желание поделиться своим мнением по поводу прочитанной книги, а также дать советы по вопросам, не свя-

занным с ошибками в листингах. Мы с радостью приветствуем такую инициативу, ведь внося свои замечания, вы избавите других читателей от многочасовых мук и, безусловно, поможете обеспечивать всех интересующихся программированием более высококачественной информацией. Вносить замечания следует при помощи ссылки **'ask a question' of our editors** (задать вопрос редактору), расположенной внизу страницы с опечатками. Щелкните по этой ссылке и перед вами появится специальный бланк, в который можно внести сообщение.

Заполните поле «тема» (subject) и напечатайте текст в окошке, предоставляемом этим бланком. Затем щелкните по кнопке **Post Now** (Отправить), расположенной внизу страницы. Сообщение будет отослано нашим редакторам. Они проверят ваше сообщение, чтобы убедиться, что ошибка действительно существует и что ваше предложение обосновано. Затем ваше сообщение, а также решение по этому вопросу будет помещено на Web-сайт для всеобщего обозрения. Безусловно, этот процесс займет день-другой, но мы прилагаем максимальные усилия, чтобы вносить исправления в как можно более короткий срок.

Поддержка по электронной почте

Если вам потребуется задать вопрос непосредственно эксперту, хорошо знакомому с книгой, то пошлите по адресу support@wrox.com электронное сообщение, в поле «тема» которого обязательно укажите название книги и последние четыре цифры номера ISBN. На рис. H5 показано, что еще следует указать в электронном письме.

Чтобы сэкономить ваше и наше время и чтобы ответ был как можно более деловым и полезным, предоставьте нам максимально подробную информацию. Если нам придется заменить вашу дискету или компакт-диск, мы сможем это сделать тотчас.

Когда кто-то из читателей посылает в наш адрес электронное сообщение, оно проходит по описанной далее цепочке.

Обслуживание клиентов

Первоначально сообщение попадает в отдел по обслуживанию клиентов. У сотрудников отдела имеется огромная база данных с ответами на наиболее часто задаваемые вопросы, так что если этот вопрос уже встречался в нашей почте, они смогут почти мгновенно отправить вам персональное сообщение.

Редакционный отдел

Более серьезные вопросы пересылаются техническим редакторам, ответственным за выпуск конкретных книг. Эти люди обладают огромным опытом работы с языками программирования, а также с многочисленными программными продуктами, поэтому они в состоянии ответить на технические вопросы. Как только поставленная проблема будет решена, редактор может поместить соответствующую информацию на Web-сайт.

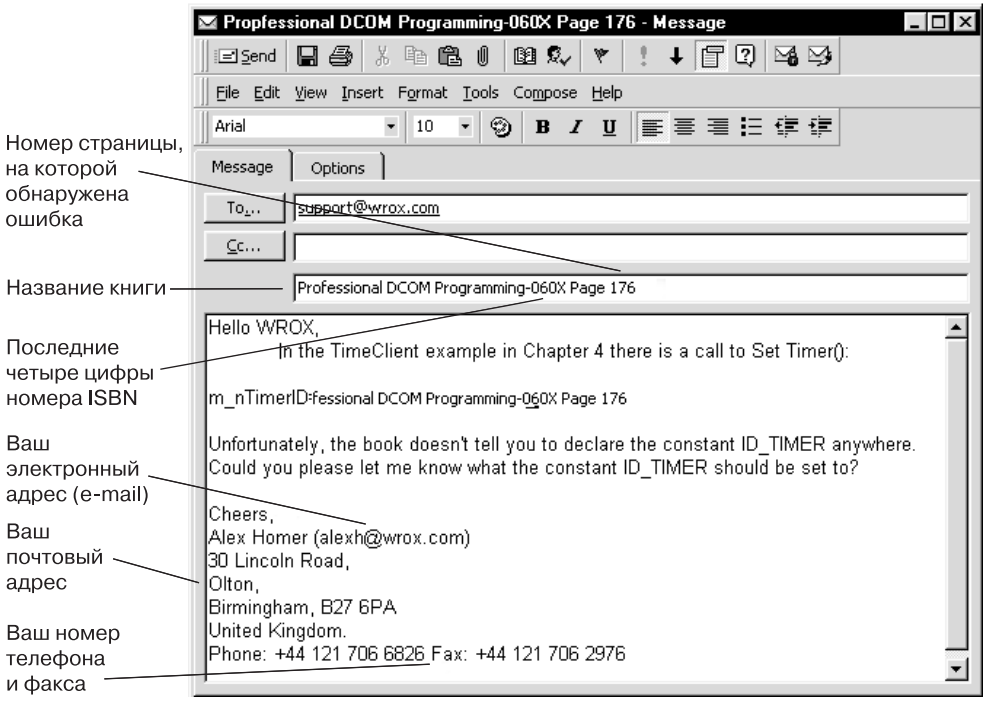


Рис. Н.5. Образец письма эксперту

Авторы

Наконец, если и редактор не в силах ответить на ваш вопрос (что мало вероятно), он перешлет ваше сообщение автору. Все авторы, сотрудничающие с издательством Wrox, рады помочь и поддержать наших читателей, так что издатели всегда готовы переадресовать им наиболее трудные вопросы. Однако следует учесть, что мы очень дорожим временем наших авторов и всеми силами стараемся не отвлекать их понапрасну. Авторы посылают свои ответы и пользователю и редактору, доступ к этой информации могут получить все желающие.

На какие вопросы мы не можем ответить

Очевидно, что с расширением тематики издаваемых книг и при постоянно меняющихся технологиях обработки информации, объем данных, требующих предоставления постоянной поддержки, многократно увеличивается. Мы сознаем свою ответственность и пытаемся ответить на все вопросы, связанные с материалом, изложенным в наших книгах, однако издательство не имеет возможности заниматься решением проблем, возникающих в программах, создаваемых отдельными читателями на основе наших кодов. Не стоит надеяться на сочувствие, если вы нанесете урон вашей компании при внедрении в жизнь программ из десятой

главы, пусть даже вам, возможно, очень пришлись по сердцу системы поддержки по Active Server Pages. Однако, если вам было особенно приятно воспользоваться алгоритмами, разработанными с нашей помощью, пожалуйста, не поленитесь проинформировать издательство.

Способ связи с издательством «ДМК»

Возможно, вам захочется сообщить нам о том впечатлении, какое произвела на вас эта книга, или вы захотите поделиться соображениями о том, как ее можно улучшить. Пожалуйста, посылайте свои сообщения по адресу info@dmk.ru. Нам важен любой ваш отзыв, отклик, сообщение, и мы постараемся сделать все возможное, чтобы должным образом ответить на него.

Алфавитный указатель

A

Б

В

Г

Д

З

И

K

Л

М

Н

О

П

P

C

T

X

Ц

Ч

Ш

Э

Я

A

Фрэнк Бумфрей, Оливия Диренцо, Джон Дакет,
Джо Грэф, Дэйв Холэндер, Пол Хоул, Тревор Дженкинс,
Питер Джоунс, Эдриан Кингсли-Хьюз, Кэти Кингсли-Хьюз,
Крэг Маккуин, Стивен Мор

XML
Новые перспективы WWW

Главный редактор *Мовчан Д. А.*
Перевод с английского *Глинка В. М.*
Выпускающий редактор *Виноградова Н. В.*
Литературный редактор *Ишков М. Н.*
Технический редактор *Прока С. В.*
Верстка *Татаринов А. Ю.*
Дизайн обложки *Антонов А. И.*

Гарнитура «Петербург». Печать офсетная.
Усл. печ. л. 43. Тираж 3000 экз. Зак. №

Издательство «ДМК», 113184, Москва, Пятницкий пер., д. 3., стр. 3.

Отпечатано в полном соответствии
с качеством предоставленных диапозитивов
в ППП «Типография «Наука»
121099, Москва, Шубинский пер., 6.