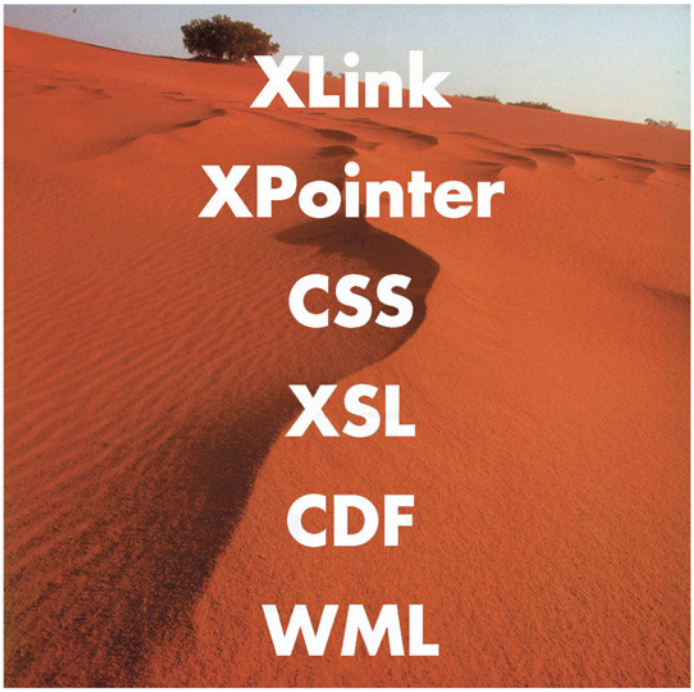
И. Шапошников

bhv
www.bhv.ru
www.bhv.kiev.ua

СПРАВОЧНИК WEB-МАСТЕРА

XML

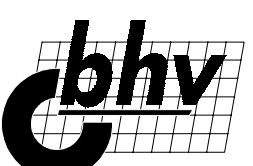
МАСТЕР



Игорь Шапошников

Справочник Web-мастера

XML



Дюссельдорф ◆ Киев ◆ Москва ◆ Санкт-Петербург

УДК 681.3.06

Справочник по современным технологиям создания и обработки документов, предназначенных для опубликования в сети Интернет, — стандарту XML и его расширениям. Приведены определения структурных элементов языка разметки XML и его синтаксис, вопросы стилевого оформления XML-документов (CSS и XSL), сведения о создании гиперссылок (XLink) и идентификации ресурсов (XPointer), о каналах CDF в Интернете и WAP-ресурсах. Описание сопровождается большим количеством примеров. Дополнительно включены официальные спецификации XML, XML Schema и WML.

Для широкого круга программистов и Web-дизайнеров

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зав. редакцией	*Наталья Таркова*
Редактор	*Евгений Васильев*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Ангелины Лужиной*
Зав. производством	*Николай Тверских*

Шапошников И. В.
Справочник Web-мастера. XML. — СПб.: БХВ-Петербург, 2001. — 304 с.: ил.

ISBN 5-94157-049-X

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 22.01.01.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 24,51.
Тираж 5000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар, № 77.99.1.953.П.950.3.99 от 01.03.1999 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с диапозитивов
в Академической типографии "Наука" РАН.
199034, Санкт-Петербург, 9-я линия, 12.

Содержание

Благодарности

Даниленко Ольге

Who wants to live forever without you?

Прежде всего, спасибо вам, что вы держите сейчас книгу в руках и читаете ее. Но поверьте мне, один я не в состоянии был ее сделать. Всегда есть люди, чей вклад в книгу является не менее весомым, чем авторский. Это, прежде всего, редакторский коллектив: главный редактор Екатерина Кондукова, с ее потрясающим знанием специфики компьютерной индустрии и чувством перспективы, заведующая редакцией Наталья Таркова, дирижировавшая всем процессом редактирования, и принимающий редактор Васильев Евгений, который приложил поистине титанические усилия для превращения текста в книгу.

Особо следует отметить корректора, на чью долю выпал поиск ошибок, пропущенных мной и моим спеллчекером. А они, поверьте мне, были.

Честно говоря, упоминать надо весь технический состав издательства "БХВ-Петербург", который принимал участие в работе над этой книгой. Спасибо вам.

Отдельное и просто огромное спасибо моей Ольге, которая постоянно поддерживала меня в работе.

Спасибо всем моим друзьям в Сети. EILA, Bellefi, Платон, Нюта, Анабелька, Ventral, Финист, Риоха и Готик — мой ящик всегда открыт для вас.

Шапошников Игорь

shiva1@yandex.ru

Введение

Абсолютное большинство всех документов в WWW написано на языке HTML (HyperText Markup Language). Но, к сожалению, на данный момент возможностей этого языка не хватает Web-мастерам для адекватного воплощения их идей. HTML-документы предназначены для отображения в браузерах, и поэтому не могут служить полноценным интерфейсом между пользователями и базами данных. Содержимое баз данных мы можем публиковать в HTML-документах, но вот обратный перенос является уже нетривиальной задачей.

Косвенным свидетельством того, что HTML исчерпал себя, может служить количество вспомогательных технологий, которые позволяют оживить Web-страницы, придать им интерактивность. Языки сценариев VBScript и JavaScript, CGI-приложения, Java-апплеты, встраиваемые модули. Какие только средства не были созданы для того, чтобы обойти ограничения HTML. Но, несмотря на ряд недостатков, HTML все равно остается сердцевиной всех технологий WWW. Мы можем пытаться обойти его ограничения, но эффективности нашей работе это не прибавит.

Исходя из этих соображений, Консорциум WWW (W3C) разработал более мощную и гибкую технологию XML (eXtensible Markup Language), призванную заменить устаревший HTML.

Данная технология на момент написания книги не является стандартом. Версия, действующая в настоящее время, является только кандидатом на стандарт (Candidate Recommendation). Однако, несмотря на то, что формально статус XML еще не определен, этой технологией уже широко пользуются Web-мастера во всем мире (например, при создании онлайновых магазинов или бюро путешествий). Активно создаются XML-приложения, многие документы преобразуются к стандарту XML. Люди сознательно идут на риск. Ведь если правила не утверждены, то многое может еще измениться.

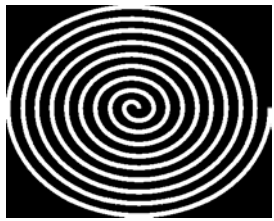
Это, кстати, объясняет ситуацию и с пособиями по XML. Очень часто в разных книгах можно встретить даже различные наборы ключевых слов и предопределенных констант. В этой книге мы не отступим ни на символ от официальной рекомендации W3C. Вы держите в руках справочник, который максимально точно и полно описывает последнюю доступную версию рассматриваемого стандарта.

В первой главе мы рассмотрим непосредственно стандарт XML, точнее — его актуальную версию, которая объявлена кандидатом на стандарт. Разберем примеры, научимся составлять собственные XML-документы. Вторая глава

посвящена обзору обособленной части стандарта — спецификации ссылок XLink. В третьей главе представлены принципы идентификации документов и их фрагментов в XML — язык XPointer. Четвертая глава содержит обзор наиболее популярного приложения XML для push-технологии — специализированного языка разметки CDF (Channel Definition Format, формат определения каналов). В следующей главе мы перейдем к рассмотрению технологии создания Web-страниц, ориентированных на доступ к ним с беспроводных тонких терминалов, то есть с сотовых телефонов. Для этих целей обычно применяется язык WML (Wireless Markup Language), который также является приложением XML. Шестая глава посвящена вопросам правильного отображения XML-документов при помощи стилевых таблиц CSS (Cascading Style Sheets). И, наконец, последняя глава ознакомит вас с преемником CSS, созданным специально для XML. В ней мы рассмотрим язык правил представления (листов стилей) XML-документов для различных устройств и сред — XSL (eXtensible Stylesheet Language).

В каждой главе излагается по возможности предельно подробная информация относительно каждой из перечисленных выше технологий. Вам предлагается описание последней, наиболее свежей версии стандарта. Весь текст в этой книге выверен, а примеры не содержат ошибок. В мир XML мы войдем вместе.

Глава 1



Расширяемый язык разметки XML

Зачем нам это надо?

Как мы уже говорили в предисловии, основой WWW является HTML (HyperText Markup Language). Этот язык представляет собой набор *тэгов* (управляющих дескрипторов), которые позволяют создавать *разметку* документа. То есть помимо указания содержимого документа, мы можем при помощи тэгов HTML управлять отображением этого документа. Технология проста. Браузер получает HTML-документ и анализирует его. Как только в коде документа встречается какой-либо тэг, он распознается, и фрагмент документа, к которому относится тэг, оформляется соответствующим образом.

На данный момент доступна уже четвертая версия языка HTML. Первоначального набора тэгов не хватало для адекватного представления более сложных документов. Поэтому компании Microsoft и Netscape — владельцы двух наиболее популярных браузеров, периодически дополняли наборы тэгов, распознаваемых их браузерами. Вследствие конкуренции этих двух фирм дополнения, естественно, не совпадали. В силу этого разработчики либо не использовали дополнительные средства, либо отказывались от возможности адекватно отображать свой документ в любом браузере. Создавался документ, предназначенный для просмотра в конкретном обозревателе, а часть посетителей сайта, использующих иной браузер, не могла увидеть страницу, содержащую данный HTML-документ (либо видела с искажениями). Проблемой HTML стала его врожденная ограниченность. Даже самый большой список тэгов не в состоянии полностью удовлетворить запросы создателей документов просто в силу того, что этот список ограничен.

Еще одной проблемой стала излишняя популярность WWW. На основе этой технологии стали строить даже локальные сети, которые, соответственно, получили название *интрасетей*. Все корпоративные документы переводились в HTML-формат. Общение внутри сети происходило при помощи электронной почты, создавались сайты, не имеющие выхода в Интернет и служащие хранилищем для корпоративных документов и средством совместной деятельности рабочих групп. Но HTML не мог служить интерфейсом между пользователями и данными. HTML-документы ориентированы прежде всего на отображение, а не на автоматическую обработку. Из базы дан-

ных можно передать данные в HTML-документ. Обратная операция намного сложнее.

Исходя из этого, стало ясно, какими свойствами должен обладать преемник HTML. Требовалось, чтобы новый язык был расширяемым, то есть не зависел от конкретного набора тэгов. Требовалась возможность расширять язык автоматически, исходя из нужд пользователя. Также было необходимо создавать файлы, которые могли бы не только отображаться, но и обрабатываться сторонними приложениями. То есть структура документов должна была напрямую привязываться к ним самим. Каждый документ должен был содержать как информацию, так и ее структуру.

Учитывая эти и многие другие соображения, Консорциум World Wide Web (W3C) в 1996 году приступил к разработке спецификаций нового языка, который должен был прийти на смену HTML. Его назвали XML (eXtensible Markup Language). На данный момент вторая редакция спецификации языка версии 1.0 является кандидатом на официальный стандарт. В этой книге мы рассмотрим данную спецификацию полностью, воспользовавшись документацией самого W3C.

Корни XML

Прежде всего, давайте разберемся, откуда появился XML. Очень давно (по меркам компьютерной индустрии, естественно), в 1986 году организацией ISO (International Organization for Standardization) язык SGML (Standard Generalized Markup Language) был принят в качестве официального стандарта. А использоваться он начал еще до этого момента. Язык SGML применяется в качестве стандарта по настоящее время. Он позволяет описывать структурированные данные, организовывать и представлять информацию, содержащуюся в документах. Стандарт SGML позволяет разработчику создавать свои конструкции разметки. Его потрясающие гибкость и универсальность, охватывающие практически все случаи, возникающие в работе над Web-проектами, казалось бы, выдвигали этот язык идеальным кандидатом для принятия его в качестве основного языка WWW, но существовали и другие обстоятельства, которые помешали ему занять лидирующее место.

Большинство документов в WWW предназначены для просмотра специализированными программами — браузерами. Браузеры анализируют код полученного документа, и на основе инструкций разметки, называемых также тэгами, отображают его в окне просмотра соответствующим образом. Но описание спецификации SGML занимает более 500 страниц текста. Отсюда видно, сколько труда потребовалось бы от разработчиков, чтобы заставить браузеры правильно отображать документы, написанные на SGML. Требовалось что-то гораздо более компактное. Естественным образом и был соз-

дан язык HTML, являющийся очень ограниченным и нерасширяемым подмножеством SGML. Так как набор тэгов HTML был невелик, разрабатывать HTML-документы и программы их просмотра было достаточно легко.

Но впоследствии это достоинство HTML превратилось в его недостаток. Посетителям и владельцам сайтов хотелось получать от HTML все больше и больше. Компании — участники "браузерных войн" добавляли все новые и новые тэги в наборы распознаваемых своими браузерами инструкций. Основная проблема состояла в том, что добавленные тэги у различных компаний тоже были разными. Возникли проблемы с совместимостью, которые не решило и доведение стандарта HTML до версии 4.0. Практически всем стало очевидно, что поскольку каждая версия HTML представляет собой ограниченный и нерасширяемый набор тэгов, то рано или поздно она окажется недостаточной.

На смену HTML пришел стандарт (а точнее, рекомендация к стандарту) XML (eXtensible Markup Language). Это — расширяемый язык разметки. Набор тэгов XML много меньше по объему, чем в HTML, но в данном случае это неважно. Изменилась сама парадигма создания документов. Появилась возможность создавать собственные тэги и конструкции из них, наподобие строительных блоков конструктора Lego. Мы теперь можем при помощи тэгов XML создавать свой язык для каждого типа документов, или даже для каждого документа отдельно.

Подобная гибкость языка позволила практически прозрачно стыковать документы с различными источниками данных для них. При помощи XML стало максимально легко интегрировать данные из различных приложений. А ведь именно интеграции различной информации из разнородных приложений подчас не хватает настоящим рабочим, а не презентационным, корпоративным сайтам. XML подоспел вовремя.

XML, как и его предшественник — HTML, является подмножеством SGML. Но XML представляет собой намного более компактный язык. Поэтому реализация браузеров для XML-документов не является излишне сложной задачей.

Любой документ из WWW в конце концов необходимо отобразить на экране компьютера удаленного пользователя. Для этой цели применяются сейчас программы-обозреватели. Браузер Internet Explorer 5-ой версии технически способен распознавать и анализировать XML-файлы, но до полного счастья еще очень далеко. Адекватное отображение содержимого XML-файлов достигается далеко не всегда.

Намного лучше дело обстоит с Netscape Communicator 5. Помимо распознавания и анализа этот браузер может еще и правильно отображать XML-документы. В него встроен полноценный XML-анализатор. Для отладки документов рекомендуется воспользоваться именно браузером Netscape.

Структура XML-документов

В XML-документах можно выделить две основные части. Первая часть XML-документа предназначена для описания его структуры, а во второй находится непосредственно содержание документа. В описании структуры документа мы можем использовать инструкции для XML-процессора, объявление элементов структуры документа, атрибуты для каждого элемента и так называемые *сущности*. Каждую из этих структурных единиц мы рассмотрим отдельно и подробно.

Описание структуры документа называют DTD-блоком (Document Type Definition). В нем мы указываем отношения на иерархии древовидной структуры элементов документа. Наиболее близкой аналогией для этих элементов могут служить объекты. Как и объекты, элементы разметки структуры могут иметь свойства, которые в данном случае называются *атрибутами*.

Как и в любой объектной иерархии, в XML-документе существует некий корневой элемент, от которого наследуются все остальные элементы.

Содержимое XML-документа форматируется при помощи тэгов, которые определяются в описании типа документа. Наименования тэгов полностью совпадают с наименованиями элементов. А параметры тэгов позволяют устанавливать значения атрибутов элементов.

Инструкции XML-процессора

В качестве первой строки каждого XML-документа должна использоваться исполняемая инструкция, предназначенная для XML-процессора (исполняемые инструкции используются для управления процессом разбора документа). В своем минимально применимом виде она выглядит следующим образом:

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

Как видно из примера, исполняемые инструкции обрамляются специальными ограничителями, состоящими из угловой скобки и знака вопроса. Ключевым словом для каждой исполняемой инструкции является сокращение `xml`. Следом за ним указываются параметры инструкции и соответствующие им значения. Иногда блок исполняемых инструкций называют *прологом* XML-документа.

Приведенный нами пример предназначен для указания браузеру, что данный документ написан на XML. Значение параметра `version` указывает на тот факт, что будет использоваться первая версия стандарта XML. (А другой версии у нас пока и нет.)

В этих инструкциях мы можем не только указывать номер версии стандарта. Для XML-документов заготовлено несколько видов кодировок, состоящих из символов набора Unicode. Большинство кодировок предложено международной организацией по стандартизации (ISO).

Для указания конкретной кодировки, которая будет использоваться для отображения содержимого документа, применяется параметр `encoding`, как в следующем далее примере:

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

В качестве значения параметра здесь задана текстовая строка `"UTF-8"`. Кодировка UTF-8 является одной из наиболее часто используемых кодировок.

Следующие параметры исполняемых инструкций XML-процессора напрямую связаны с обработкой блоков описания структуры документа (DTD-блоков). Забегая немного вперед, необходимо сказать, что подобные блоки могут внедряться в сам XML-документ, или находиться во внешнем файле. Для того чтобы XML-процессор смог правильно их обработать, применяется параметр `standalone`. При помощи этого параметра можно указать, где именно находится описание структуры для данного XML-документа. В следующем примере мы используем объявление XML-документа, в котором указывается, что DTD-блок для него оформлен в виде отдельного файла.

```
<?xml standalone='no'?>
```

У параметра `standalone` есть два предопределенных значения: `yes` и `no`. Значение `no`, как мы уже видели, извещает XML-процессор, что для данного документа DTD-блок выделен в отдельный файл. Значение `yes` указывает на то, что DTD-блок размещен в теле XML-файла.

Мы рассмотрели возможные варианты исполняемых инструкций, помещаемых в начале документа. Практически эти инструкции могут находиться не только в начале документа, но и в любом другом его месте. Но использование исполняемой инструкции в качестве первой строки XML-документа является обязательным условием.

Любое рассмотрение языка на примерах, без четких определений, будет всего лишь обзором. Чтобы избежать этой участи, для каждой из рассматриваемых нами структурных единиц XML-документа мы будем приводить их определение из спецификации XML. Для исполняемых инструкций, помещаемых в пролог документа, оно выглядит следующим образом:

```
[23]  XMLDecl ::=  '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'
[24]  VersionInfo ::=  S 'version' Eq (' VersionNum ' | " VersionNum ")
[25]  Eq ::=  S? '=' S?
[26]  VersionNum ::=  ([a-zA-Z0-9_.:] | '-')+
```

На первый взгляд смотрится достаточно устрашающе. Тем не менее, все спецификации XML выглядят подобным образом. В примере приведена запись конструкций XML в форме Бэкуса-Наура (EBNF, Extended Backus-Naur Form). Разберемся с правилами чтения деклараций в нотации EBNF. В левой части каждой декларации указывается имя конструкции, затем следует оператор эквивалентности (::=), а в правой части следует расшифровка имени, которая содержит его формат и правила оформления. Перед каждой конструкцией в квадратных скобках указывается номер строки спецификации.

Итак, из приведенного фрагмента мы видим, что определение исполняемой инструкции находится в 23-й строке спецификации языка XML. При изучении спецификации необходимо помнить несколько правил записи определений в форме EBNF.

- Если в определение включено несколько терминов, разделенных вертикальной чертой (|), то следует выбрать только один из них. Подобным образом определяется перечислимое множество компонентов.
- Для указания диапазона символов, которые могут использоваться в каком-либо месте определения термина, этот диапазон символов помещается в квадратные скобки.
- Круглые скобки ограничивают так называемые *регулярные выражения*. Проще всего их воспринимать как обычное средство группировки элементов. В случае, когда согласно синтаксису вместо одиночного литерала нужно указать несколько, используются круглые скобки, группирующие их.
- Знак звездочки (*) примыкает справа к символьному выражению, если требуется, чтобы это выражение в результирующей строке могло быть повторено несколько раз или вовсе там не использоваться.
- Знак плюса (+) применяется в качестве модификатора символьного выражения подобно знаку звездочки. Но есть некоторое отличие. Символьное выражение, после которого присутствует этот модификатор, должно встречаться в результирующей строке хотя бы один раз.
- Вопросительный знак (?) используется для указания, что тот или иной элемент могут не найтись в результирующей строке. То есть при помощи этого модификатора указываются опциональные элементы.
- Если в выражении не должны встречаться некоторые символы, то к нему добавляется последовательность [^, после которой указываются исключаемые символы. Например, правило [^&] исключает символ амперсанда.
- Если внутри какого-либо литерала необходимо вставить пробел, то этот пробел обозначается при помощи символа S.
- Любая последовательность символов, заключенная в двойные или одиночные кавычки, является литералом, и в результирующей строке должна появляться именно в том виде, в каком она указана в декларации.

Зная эти правила, рассмотрим синтаксис деклараций из вышеприведенного примера. Начнем, естественно, с директивы [23]. Начинает определение элемент `XMLDecl`, который и является объявлением XML-документа. После оператора эквивалентности указан литерал `'<?xml'`, означающий, что строка объявления XML документа должна начаться с открывающей угловой скобки, вопросительного знака и последовательности символов `xml`. Впрочем, этот факт мы выяснили еще до разбора синтаксиса декларации. Следующим элементом декларации является обязательный элемент `VersionInfo`, указывающий номер версии применяемого стандарта XML. Помимо этого в объявлении могут присутствовать конструкции `EncodingDecl` и `SDDecl`. Завершается декларация последовательностью символов `'?>'`, перед которой может быть размещен необязательный пробел.

В приведенном выше фрагменте спецификации XML отсутствуетопределение конструкции `SDDecl`. Приведем его сейчас:

```
[32]  SDDecl ::=  S 'standalone' Eq (("'" ('yes' | 'no') "'") |
('"' ('yes' | 'no') '"'))
```

Как видно из этой декларации, значения параметра `standalone` (использование внешних DTD-описаний) могут заключаться как в двойные, так и в одиночные кавычки.

Объявление типа документа

Сразу после исполняемой инструкции, указывающей на тот факт, что данный документ создан с применением языка XML, в этом документе помещается объявление типа документа, называемое также DTD (Document Type Definition). DTD-блок является основой для структуризации содержимого XML-файла. То есть этот блок включает в себя правила, по которым происходит разметка содержимого документа. Здесь определяются элементы документа, атрибуты для этих элементов, сущности и комментарии. Фактически DTD-блок не менее важен для XML-документа, чем его значимое содержимое. Рассмотрим пример объявления типа документа:

```
<!DOCTYPE body [
<!ELEMENT body (#PCDATA)>
<!ENTITY name "Igor">
]>
```

В первой строке примера начинается объявление типа документа. Нетрудно заметить, что DTD-блок начинается ключевым словом `DOCTYPE`, помещаемым после открывающей угловой скобки и восклицательного знака. (Необходимо отметить, что все конструкции XML помещаются в угловые скобки. Но перед большинством ключевых слов ставится восклицательный

знак. А исполняемые инструкции XML-процессора, как мы помним, предваряются вопросительным знаком.)

После ключевого слова `DOCTYPE` указывается наименование типа. Это, по сути, имя корневого элемента объектной иерархии данного документа. Здесь необходимо сделать некоторое отступление, и рассказать немного об *элементах* XML-документа, которые более полно мы будем рассматривать уже в следующем разделе. Но в XML все настолько переплетено и взаимосвязано, что есть только один путь строго последовательного и непротиворечивого изложения правил оформления XML-документов. Это — следование официальной спецификации, фрагменты которой мы приводим в тексте этой книги. Но поскольку изучать XML по спецификации трудно, нам придется немного менять местами изложение различных структурных единиц, принося последовательность изложения в жертву его доступности.

Итак, элементы в XML являются основными структурными единицами. Их ближайшей аналогией являются тэги HTML, посредством которых мы могли создавать абзацы, элементы списков, ссылки и прочие элементы разметки. Безусловно их можно назвать прародителями элементов XML. Но в HTML элементы разметки не были взаимосвязаны. В XML же был сделан качественный шаг вперед. Теперь элементы могут быть структурированы. Все элементы зависят друг от друга. В XML-документе обязателен основной элемент, составляющими которого являются другие элементы. То есть несколько упрощенно моделируется объектная иерархия, в которой в качестве объектов выступают элементы XML. Как мы увидим немного позже, это — достаточно хорошая аналогия, и мы еще не раз воспользуемся ею.

Таким образом, в качестве наименования типа XML-документа мы используем имя самого старшего элемента, который включает в себя все остальные элементы. В нашем примере это элемент с именем `body`.

После указания этого имени в квадратных скобках описывается структура всего документа. А после закрывающей квадратной скобки следует закрывающая угловая скобка. Получается, что все определение типа документа находится в рамках одного тэга `DOCTYPE`. Все эти признаки нетрудно обнаружить в коде рассмотренного нами примера.

DTD-блоки могут находиться как внутри XML-документа, так и вне его. Для нескольких XML-документов можно создать один файл с описанием DTD, и использовать его для каждого из этих XML-документов. Для подключения внешнего DTD-блока можно использовать конструкцию, подобную следующей:

```
<!DOCTYPE main PUBLIC "-//New Boundaries//Sweet Immersing//EN"
↳"http://www.newboundaries.com/sweet/dtd/main.dtd">
```

Разберем этот пример. Мы объявляем тип документа `main`, спецификация которого находится в отдельном DTD-файле. В данном случае объявлено,

что это DTD-описание не было принято международной организацией по стандартизации (ISO) в качестве стандарта, оно принадлежит компании New Boundaries, создано для проекта Sweet Immersing, ориентировано на англоязычные документы, и файл со спецификацией находится по адресу `http://www.newboundaries.com/sweet/dtd/main.dtd`.

Ключевое слово `PUBLIC` применяется для так называемых "публичных" DTD. Для них помимо URL указывают еще и наименование. Делается это для того, чтобы XML-браузер мог отыскать эти DTD на других серверах, которые, может быть, будут ближе. Конечно, в критериях Интернета понятие "ближе" относится к географии весьма опосредованно, но выбирается всегда тот DTD-файл, который будет быстрее всего загружен на локальную систему удаленного пользователя. Публичные DTD отличаются от обычных обособленных DTD-файлов тем, что могут находиться на нескольких различных серверах.

Еще одна особенность публичных DTD состоит в том, что их рассматривает международная организация по стандартизации (ISO). В том случае, если этот DTD-блок принят в качестве стандарта, перед именем файла (а не перед его URL) ставится префикс `ISO`. Если в качестве стандарта этот DTD не принят, но, тем не менее, рассматривается группой стандартизации, в качестве префикса используется знак плюса (`+`). Если же он не принят группой стандартизации, применяется префикс в виде минуса (`-`), как мы это видели в нашем примере.

Если же DTD-блок не отправлялся в ISO, то он объявляется "приватным". В таком случае вместо ключевого слова `PUBLIC` указывается ключевое слово `SYSTEM`. При этом в описании используется конструкция следующего вида:

```
<!DOCTYPE main SYSTEM "http://www.newboundaries.com/sweet/main.dtd">
```

Теперь, когда мы знаем, как использовать внешние файлы, содержащие DTD-блоки, следует научиться создавать эти файлы. Любой DTD-файл подчиняется спецификациям XML. То есть, по сути, он является обычным XML-документом, но без значимого содержимого. Поэтому первой строкой здесь также будет исполняемая инструкция XML-процессора. Но в данном случае совсем необязательно указывать номер версии стандарта XML. Обычно во внешних DTD-файлах указывают используемую кодировку, после чего определяются структурные элементы. Так как ссылка на DTD-файл встраивается в соответствующий XML-документ, нет нужды в DTD-файле использовать объявление типа документа с ключевым словом `DOCTYPE`.

Мы уже рассматривали параметр `standalone` для исполняемой инструкции, которая объявляет использование XML. В том случае, если используется внешний DTD-файл, в качестве значения параметра необходимо использовать значение "`no`".

В зависимости от того, как используется DTD-блок, какие правила оформления DTD применяются в XML-документах, эти документы могут называться "хорошо оформленными" (well-formed) и "правильными" (valid). Кстати, некоторые правила спецификации XML являются обязательными для оформления документа, если мы хотим, чтобы этот документ являлся хорошо оформленным или правильным. Для таких правил в спецификации указываются индексы `WFC` (Well-Formedness Constraint) и `VC` (Validity Constraint) соответственно. Для того чтобы избавить вас от штудирования неудобочитаемой спецификации, изложим здесь эти правила более понятным языком.

Итак, для того чтобы документ считался *хорошо оформленным*, необходимо выполнение нескольких правил.

- У документа должен быть только один элемент верхнего уровня, и содержимое документа должно полностью располагаться внутри тэга, соответствующего этому элементу-прародителю.
- В одном тэге не могут употребляться несколько раз одни и те же атрибуты элемента, соответствующего этому тэгу.
- Все сущности должны объявляться до их использования.
- Все тэги должны быть правильно вложены друг в друга, согласно иерархии элементов, каждый открывающий тэг должен иметь своего закрывающего "напарника" (нельзя опускать закрывающие тэги).

Вне сомнения, все XML-документы, которые будут использоваться в публичных целях, просматриваться в различных браузерах и обрабатываться различными приложениями, должны быть, как минимум, хорошо оформленными. Обязательное использование вышеперечисленных правил обусловлено тем, что при несоблюдении какого-либо из них обычный XML-процессор выдаст сообщение о фатальной ошибке, после чего приложение, в котором действует XML-процессор, будет остановлено. Более того, кое-где встречается информация о том, что отдельные приложения и браузеры "зависают" при попытке просмотра документов с ошибками.

Следует обратить внимание на то, что в перечне правил для хорошо оформленных документов нет ни слова о DTD-блоках. Связано это с тем, что согласно спецификации DTD-блоки не являются обязательными для использования в XML-документах. Следующий XML-документ считается правильным, так как не нарушено ни одно из правил:

```
<?xml version="1.0" standalone="yes"?>
<body>well-formed document </body>
```

Следует также учитывать, что XML-процессоры чувствительны к регистру символов.

Правильным считается XML-документ, который удовлетворяет требованиям, предъявляемым к хорошо оформленным документам, и при этом имеет соответствующий DTD-блок и подчиняется всем правилам, описанным в нем.

Для того чтобы вышеприведенный пример XML-документа можно было бы считать правильным, необходимо внести в код примера некоторые изменения, после чего он будет выглядеть следующим образом:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE body [<!ELEMENT body (#PCDATA)>
]>
<body>valid document </body>
```

Теперь же, после рассмотрения примеров DTD-блоков и всех правил, связанных с ними, приведем фрагмент спецификации, описывающий определение типа документа:

```
[28]  doctypedecl ::=  '<!DOCTYPE' S Name (S ExternalID)? S?
      ↳('['(markupdecl | PEReference | S)* ']' S?)? '>'
      [VC: Root Element Type ]
[29]  markupdecl ::=  elementdecl | AttlistDecl | EntityDecl
      ↳| NotationDecl | PI | Comment
      [VC: Proper Declaration/PE Nesting ]
      [WFC: PEs in Internal Subset ]
```

Здесь видно, каким образом в спецификации описывается дополнительное условие наличия корневого элемента, необходимое для создания правильного документа.

Элементы XML-документа

В этой главе мы уже говорили об элементах. Напомним, что элементы являются основными структурными единицами XML-документов. Мы объявляем все элементы документа в DTD-блоке, а потом при разметке значимого содержимого документа мы используем тэги, наименования которых совпадают с наименованиями элементов.

Объявление элемента в самом упрощенном виде мы уже наблюдали в одном из примеров, но сути пока не раскрывали. Приведем текст примера еще раз:

```
<!DOCTYPE body [
<!ELEMENT body (#PCDATA)>
]>
```

Здесь мы объявляем тип документа `body`, для которого, в свою очередь, объявляется одноименный элемент. Как видно, объявление элемента производится при помощи ключевого слова `ELEMENT`, после которого указывается наименование элемента и его тип. Все значимые и обязательные части инструкции разделяются пробелами. В данном случае мы объявили элемент с символьным содержимым, применив для этого стандартную конструкцию `(#PCDATA)` (что означает `parseable character data` — любая информация, с которой может работать XML-процессор).

На самом деле элементы в качестве своего содержимого могут использовать не только символьные данные. Элементы могут содержать другие элементы, которые, в свою очередь, могут тоже содержать элементы, и так далее. То есть выстраивается некая иерархия элементов, вложенных по типу матрешки в основной элемент, объявленный как тип документа при помощи ключевого слова `DOCTYPE`.

Рассмотрим пример такой усложненной организации. Предположим, что мы создаем XML-документы для некоей компании, которая поддерживает базу данных предприятий города. Нам потребуется создать основной элемент `firm`, к которому будут привязаны вложенные элементы, отражающие название фирмы, ее род деятельности, адрес, телефоны, сетевые реквизиты и т. д. Такую организацию можно проиллюстрировать следующим примером:

```
<!ELEMENT firm (name+, address, phone*, fax*, email, info)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT fax (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT info (#PCDATA)>
```

В этом примере мы создаем элемент, в состав которого входят шесть других элементов с содержимым символьного типа. При этом для группировки субэлементов используются в соответствии с синтаксисом круглые скобки. Все, казалось бы, достаточно прозрачно, но если приглядеться, видно, что после наименований некоторых вложенных элементов установлены дополнительные модифицирующие знаки. Назовем их *модификаторами*. Всего в XML применяется три модификатора.

- Знак плюса (`+`) применяется для таких элементов, наличие которых в родительском элементе является обязательным. Данные элементы могут встречаться в рамках родительского элемента несколько раз. Но, по крайней мере один раз они должны быть задействованы обязательно.
- Знак звездочки (`*`) указывает на то, что данный элемент может встречаться в описании родительского элемента любое количество раз. Впрочем, может там вообще не присутствовать.
- Вопросительный знак (`?`) используется для тех элементов, которые могут появляться в описании родительского элемента только один раз, или вообще не появляться.

Все остальные символы, которые могут встречаться в описании типа элемента, и в то же время не входят в наименования субэлементов, служат либо для группировки, либо в качестве разделителей.

Группировка элементов, как мы знаем, осуществляется при помощи круглых скобок. При помощи же запятой мы фактически устанавливаем порядок следования элементов. То есть в значимом содержимом XML-документа все субэлементы должны следовать именно в том порядке, в каком они перечислены в DTD-блоке.

В том случае, когда необходимо использовать только один из некоторого множества субэлементов, это множество дополнительно заключается в круглые скобки, и все наименования субэлементов разделяются вертикальной чертой (|). Например, если бы мы хотели, чтобы в нашем примере для элемента `firm` можно было бы использовать в качестве реквизита либо номер факса, либо номер телефона, мы могли бы написать следующую декларацию:

```
<!ELEMENT firm (name+, address, (phone | fax), email, info)>
```

Элементы не обязательно должны состоять из других элементов. В любой иерархии должны найтись такие элементы, которые не будут содержать дочерних элементов. Подобные элементы называются *атомарными*.

Это, впрочем, не означает, что в атомарных элементах будут находиться единичные и неделимые фрагменты информации. Каждый элемент может обладать *атрибутами*, которые в свою очередь и будут содержать подобные информационные атомы. Хорошей естественной аналогией для атрибутов являются свойства объектов. (В этой части мы рассматриваем только элементы. Рассмотрение же атрибутов отложим до следующего раздела.)

Итак, нас интересует, как задавать тип для атомарных элементов. На выбор имеются три варианта.

- Если мы собираемся сохранять в элементе какие-либо данные так называемого *смешанного типа* (mixed content), мы должны указать в круглых скобках тип `#PCDATA`. По сути, под эвфемизмом "смешанный тип" подразумевается использование любых символьных данных.
- Если заранее не определено, какой тип данных будет содержать этот элемент, то для указания его типа используется ключевое слово `ANY`.
- Для обозначения типа *пустых* элементов (заглушки) используется ключевое слово `EMPTY`.

Все типы могут комбинироваться в объявлении элемента различным образом для достижения именно того результата, который нам необходим. В качестве примера можно рассмотреть следующие декларации:

```
<!ELEMENT b (#PCDATA)>
<!ELEMENT br EMPTY>
<!ELEMENT container ANY>
<!ELEMENT p (#PCDATA|a|ul|b|i|em)*>
```

Это определение имеет следующий смысл: элемент `b` содержит символьные данные (используется, так называемый *смешанный* тип содержимого), эле-

мент br изначально задан пустым, элемент container предназначен для хранения данных неопределенного типа, а элемент p может содержать либо символьные данные, либо один из других элементов, наименования которых указаны в скобках и разделенны вертикальной чертой.

И в заключение обзора элементов в соответствии с нашими правилами мы должны привести фрагмент спецификации, описывающий объявление элементов XML-документа.

```
[45]  elementdecl ::=  '<!ELEMENT' S Name S contentspec S? '>'
      [ VC: Unique Element Type Declaration ]
[46]  contentspec ::=  'EMPTY' | 'ANY' | Mixed | children
[47]  children ::=  (choice | seq) ('?' | '*' | '+')?
[48]  cp ::=  (Name | choice | seq) ('?' | '*' | '+')?
[49]  choice ::=  '(' S? cp ( S? '|' S? cp )* S? ')'
      [ VC: Proper Group/PE Nesting ]
[50]  seq ::=  '(' S? cp ( S? ',' S? cp )* S? ')'
      [ VC: Proper Group/PE Nesting ]
```

Атрибуты элементов

В предыдущем разделе мы уже говорили о том, что неделимые порции информации записываются и хранятся в атрибутах элементов. При сравнении элементной модели с объектной иерархией атрибуты можно считать свойствами объектов. При помощи атрибутов мы можем максимально полно детализировать информацию, предназначенную для отображения или использования в данном элементе.

Список атрибутов задается при помощи ключевого слова ATTLIST. Рассмотрим маленький пример:

```
<!ELEMENT text (#PCDATA)>
<!ATTLIST text
       create_date   CDATA #REQUIRED
       last_modified CDATA #IMPLIED
       author        CDATA #REQUIRED
       editor        CDATA "Kondukova E.V.">
```

В этом примере мы объявили элемент text, которому присвоили четыре атрибута. Как видно, после ключевого слова ATTLIST указывается наименование элемента, к которому присоединяются атрибуты, определяемые в данном тэге ниже. После наименования элемента записывается список атрибутов. Для каждого атрибута указывается наименование, тип содержимого атрибута и модификатор атрибута.

Что касается наименования, то с ним все ясно. Наименования атрибутов полностью подчиняются правилам формирования имен в XML. Перечислим их. Имя обязано начинаться с буквенного символа, символа подчеркивания или двоеточия, и может содержать в себе любые буквы, цифры и знаки подчеркивания, двоеточия, дефиса и точки.

После имени атрибута указывается его тип. Всего может быть три типа. В нашем примере мы объявили атрибуты только символьного типа. Для обозначения этого типа применяется ключевое слово `CDATA`. В значениях этих атрибутов могут храниться данные только в виде строк.

Атрибуты могут также иметь *перечислимый* тип. Для подобных атрибутов при их объявлении заранее указывается список возможных значений, заключаемых в круглые скобки и разделяемых вертикальной чертой. В качестве примера можно привести следующее объявление атрибута:

```
<!ATTLIST my_element
        type (a|b|c) "c">
```

В этом примере объявляется атрибут `type`, который может принимать только значения `a`, `b` и `c`, причем по умолчанию используется значение `c`.

В XML применяется также и третий тип атрибутов. Атрибуты этого типа называются *маркерами*. Это — специализированные атрибуты, значения которых несут заранее предопределенный тип информации об элементе. При использовании маркеров в качестве типа атрибута мы должны включить в описание одно из семи ключевых слов, идентифицирующих вид маркера. Так, например, в последней версии стандарта HTML в каждом тэге присутствует дополнительный параметр `ID`, при помощи которого задают идентификатор блока, обрабатываемого данным тэгом. В XML этот параметр заложен изначально и является атрибутом-маркером. Приведем пример объявления подобного атрибута:

```
<!ATTLIST my_element
        id ID #REQUIRED>
```

Данный пример демонстрирует присвоение элементу `my_element` атрибута `id`, имеющего предопределенный тип `ID`.

Ниже рассмотрим все семь упомянутых ранее предопределенных типов атрибутов-маркеров.

- Атрибут типа `ID` предназначен для указания уникального идентификатора данного элемента. Значение этого атрибута должно полностью удовлетворять соглашению об именах в XML. Естественно, идентификаторы всех экземпляров элементов в содержимом XML-документа должны быть разными, иначе нарушается условие уникальности. Если же это условие будет нарушено, XML-процессор укажет на ошибку в документе.

- Атрибут IDREF содержит идентификатор другого элемента, с которым данный элемент связан по смыслу. Атрибут данного типа предназначен для реализации двунаправленных ссылок — одной из наиболее широко разрекламированных возможностей, вошедших в стандарт XML версии 1.0. Конечно, элемент, идентификатор которого указан в данном атрибуте, должен присутствовать в XML-документе, иначе XML-процессор объявит о недействительности данного документа.
- Если данный элемент семантически связан с несколькими другими документами, то мы можем указать все их идентификаторы сразу в одном атрибуте. Для этого используется атрибут типа IDREFS. В его значении все идентификаторы связанных элементов перечисляются, разделенные пробелами.
- Атрибут ENTITY указывает на внешнюю сущность, связанную с данным элементом. В качестве значения этого атрибута указывается имя сущности. Подробно механизм сущностей мы рассмотрим в следующей части.
- Атрибут ENTITIES весьма похож на предыдущий. Разница лишь в том, что в его значении мы можем записать сразу несколько имен внешних сущностей.
- Атрибут NMTOKEN содержит любую последовательность символов из имени данного элемента.
- Идею предыдущего атрибута расширяет атрибут с именем NMTOKENS, который в своем значении может содержать сразу несколько подстрок имени данного элемента.

Мы рассмотрели возможные идентификаторы типов атрибута, размещаемые в объявлениях сразу после наименования. Осталось объяснить смысл ключевых слов, завершающих определение атрибута. Мы можем задать атрибут, который будет обязателен для применения в каждом экземпляре элемента, задать значение по умолчанию для каждого атрибута, и т. д. Всего имеется три стандартных модификатора для атрибутов.

- Модификатор #IMPLIED указывает на то, что для данного атрибута значение может быть не определено, то есть этот атрибут не является обязательным для применения.
- Модификатор #REQUIRED применяется, наоборот, для указания обязательных атрибутов. То есть, если у данного атрибута некоего элемента указан этот модификатор, то во всех экземплярах элемента в значимом содержимом XML-документа должно быть обязательно присвоено значение этому атрибуту.
- Модификатор #FIXED применяется в тех случаях, когда заданное для атрибута значение по умолчанию является фиксированным, то есть не поддается изменению. Проще говоря, атрибут должен иметь только указан-

ное значение. Понятно, что этот модификатор не применяется для атрибутов перечислимого типа.

Для каждого атрибута можно задать значение, применяемое по умолчанию. Причем оно может как замещать стандартный модификатор, так и использоваться совместно с ним. Рассмотрим еще раз наш пример.

```
<!ELEMENT text (#PCDATA)>
<!ATTLIST text
       create_date   CDATA #REQUIRED
       last_modified CDATA #IMPLIED
       author        CDATA #REQUIRED
       editor        CDATA "Kondukova E.V.">
```

Теперь нам видно, что атрибуты `create_date` и `author` являются обязательными, а атрибут `last_modified` может и не использоваться в элементах `text`. Для атрибута же `editor` предусмотрено значение по умолчанию `Kondukova E.V.`, ограниченное двойными кавычками, как строковая переменная. В этом примере значение по умолчанию не комбинируется с модификатором. Пример сочетания модификатора и значения по умолчанию может выглядеть следующим образом:

```
<!ATTLIST hyperlink
↳xlink:type CDATA #FIXED "simple">
```

В этом примере мы объявляем элемент `hyperlink` с атрибутом `xlink:type`, для которого установлено фиксированное значение `simple`. Следовательно, каждый раз, когда в тексте XML-документа будет использоваться элемент `hyperlink`, то его атрибут `xlink:type` будет всегда иметь значение `simple`, и изменить это значение в тексте XML-документа для какого-либо экземпляра элемента будет нельзя.

Атрибуты перечислимого типа мы можем комбинировать со списком условных обозначений (`NOTATION`). Механизм подобных списков мы будем рассматривать в одной из следующих частей. А сейчас нам необходимо просто привести пример подобного сочетания. После наименования атрибута мы указываем в качестве его типа ключевое слово `NOTATION`. Затем (как обычно в скобках) мы указываем сами условные обозначения, составляющие список. А после перечисления мы вписываем значение, используемое по умолчанию. Такой подход демонстрируется в следующем примере:

```
<!ATTLIST new_element
↳attr NOTATION (first|second|third) ”third”>
```

Естественно, в DTD-блоке должно присутствовать определение данного списка и всех определенных в нем элементов.

Итак, мы рассмотрели все правила создания атрибутов. Напоследок приведем фрагмент спецификации, описывающей создание атрибута:

```
[52]  AttlistDecl ::=  '<!ATTLIST' S Name AttDef* S? '>'
[53]  AttDef ::=  S Name S AttType S DefaultDecl
[54]  AttType ::=  StringType | TokenizedType | EnumeratedType
[55]  StringType ::=  'CDATA'
[56]  TokenizedType ::=  'ID' [ VC: ID ]
      [ VC: One ID per Element Type ]
      [ VC: ID Attribute Default ]
      | 'IDREF' [ VC: IDREF ]
      | 'IDREFS' [ VC: IDREF ]
      | 'ENTITY' [ VC: Entity Name ]
      | 'ENTITIES' [ VC: Entity Name ]
      | 'NMTOKEN' [ VC: Name Token ]
      | 'NMTOKENS' [ VC: Name Token ]
[57]  EnumeratedType ::=  NotationType | Enumeration
[58]  NotationType ::=  'NOTATION' S '(' S? Name (S? '|' S? Name)* S? ')'
      [ VC: Notation Attributes ]
[59]  Enumeration ::=  '(' S? Nmtoken (S? '|' S? Nmtoken)* S? ')'
      [ VC: Enumeration ]
[60]  DefaultDecl ::=  '#REQUIRED' | '#IMPLIED'
      ↳| (('#FIXED' S)? AttValue) [ VC: Required Attribute ]
      [ VC: Attribute Default Legal ]
      [ WFC: No < in Attribute Values ]
      [ VC: Fixed Attribute Default ]
```

Сущности

Сущности являются одним из ключевых понятий XML. Под ними обычно подразумевают единые и неделимые блоки информации, не анализируемые XML-процессором. Сущности могут представлять собой файлы с бинарной информацией, такой как рисунки, аудио- и видеофайлы, документы, обрабатываемые специализированными приложениями, и многое другое. Также в качестве сущностей используются блоки часто обновляемой информации. Достаточно такой блок объявить в качестве сущности в DTD-описании, и потом во всех XML-документах, подключенных к этому DTD-блоку, можно применять обозначение этой сущности. Таким образом, при необходимости единовременного изменения какого-либо часто применяемого элемента во всей совокупности документов достаточно изменить его в одном месте — в объявлении сущности, и изменения во все связанные с этим DTD-блоком

XML-документы будут внесены автоматически. Также сущности могут применяться и в самом DTD-блоке для оформления часто встречающихся последовательностей атрибутов или элементов, то есть выполнять обычную функцию текстовой подстановки. При помощи все тех же сущностей мы можем вставлять в текст XML-документа неотображаемые символы. Обобщая, можно сказать, что любая сущность является контейнером для каких-либо данных, помещаемых в XML-документ. Все эти возможности мы тщательно рассмотрим и приведем соответствующие примеры.

Итак, официально сущности являются особого рода элементами разметки, которые содержат объявление текста или данных, вставляемых в значимое содержимое XML-документа при помощи связанных с ними псевдонимов.

Сущности по своему типу делятся на текстовые, бинарные и параметрические. А по месту определения (декларирования) мы можем сущности делить на внешние и внутренние.

Любая сущность объявляется в DTD-блоке при помощи ключевого слова `ENTITY`. После него, отделенное пробелом, следует наименование сущности — ее *псевдоним*. А затем, уже в самом конце декларации мы вписываем значение сущности, заключенное в двойные кавычки. В качестве примера наиболее простой текстовой сущности приведем следующую конструкцию:

```
<!ENTITY cp "copyright disclaimer">
```

Теперь, если в тексте XML-документа мы вставим наименование этой сущности, обрамленное знаками амперсанда и точки с запятой, то оно при отображении документа браузером будет заменено на соответствующее словосочетание. То есть в тексте для вызова сущности `cp` мы должны использовать конструкцию `&cp;`.

Также существуют и предопределенные текстовые сущности. В значимом содержимом XML-документа мы можем применять только символы из набора ASCII. Все остальные символы могут быть вставлены в текст только при помощи сущностей. По существу, все кодировки представляют собой набор сущностей.

В ASCII-наборе есть несколько символов, которые не могут быть в явном виде вставлены в текст XML-документа. Это — служебные символы, применяемые в разметке кода XML-документа, такие как знаки "больше" и "меньше", знак амперсанда (`&`), а также одинарные и двойные кавычки. Для вставки этих символов в текст значимого содержимого XML-документа используются сущности `>`, `<`, `&`, `&apos;` и `&quot;` соответственно.

Бинарные сущности позволяют встраивать в текст XML-документа любые внешние файлы, от графических изображений до файлов Microsoft Office (не исключаются, естественно, и мультимедиа-файлы). В общем случае к XML-документу можно присоединить любые файлы, необходимо лишь связать

эти файлы с приложениями, способными их обрабатывать. Приведем пример сущности, которая подключает к документу графический GIF-файл:

```
<!ENTITY picture SYSTEM "images/pic1.gif" NDATA gif>
```

По определению, все бинарные сущности являются *внешними*, так как они встраивают внешние по отношению к XML-документу файлы. Поэтому после объявления имени, так называемого *псевдонима* сущности, следует ключевое слово `SYSTEM`, за которым указывается URL подключаемого файла. Эти правила достаточно прозрачны, так как не противоречат ничему, что мы узнали о разметке XML ранее. Но есть одно достаточно тонкое место. XML-процессор не знает о том, какой именно тип файла мы используем. Он не обязан автоматически распознавать тип файла по расширению. Поэтому для каждого типа файлов, применяемого в оформлении документов, необходимо уточнить программу, применяемую для обработки данного типа файлов. Для этих целей применяется механизм условных обозначений (`NOTATION`), который мы детально рассмотрим в следующей части. Здесь же мы приведем только маленький пример. Если мы в объявлении сущности `picture` использовали условное обозначение `gif`, указав его после ключевого слова `NDATA`, то в DTD-блоке должно присутствовать объявление для данного типа, подобное следующему:

```
<!NOTATION gif SYSTEM "http://www.mysite.ru/progs/gifview.exe">
```

В этом объявлении мы указываем, что для обработки файлов типа `gif` необходимо вызвать приложение, URL которого указан после ключевого слова `SYSTEM` в двойных кавычках.

В объявлении бинарной сущности необходимо внимательно следить за типом файла, указываемого после ключевого слова `NDATA`. Каким бы ни было расширение подключаемого файла, XML-процессор не будет принимать его во внимание. Тип устанавливается по соответствующему обозначению.

Однако есть одна возможность создавать и использовать бинарные внешние сущности без указания типа присоединяемого файла. Это возможно, если вставляются сторонние XML-файлы. При помощи этого средства мы можем собирать объемлющий XML-документ из других документов. Например, если у нас уже есть три XML-файла, содержимое которых необходимо объединить в одно целое, мы можем объявить их как внешние бинарные сущности, и вызвать в основном документе через их псевдонимы. Приведем соответствующий пример:

```
<!xml version="1.0">
<!DOCTYPE body [
<!ENTITY doc1 SYSTEM "http://www.parts.ru/part1.xml">
<!ENTITY doc2 SYSTEM "http://www.parts.ru/part2.xml">
<!ENTITY doc3 SYSTEM "http://www.parts.ru/part3.xml">
]>
```

При этом в теле документа следует использовать следующую конструкцию:

```
<body>
&doc1;
&doc2;
&doc3;
</body>
```

Тем не менее мы можем использовать в объявлении внешней бинарной сущности не только ключевое слово SYSTEM, но и PUBLIC. Но в этом случае мы должны подставлять только те файлы, которые рассматривались международной организацией по стандартизации. Хотя с другой стороны, так ли уж необходимо вам стандартизовывать свои данные?

Нам осталось рассмотреть третий вид сущностей — *параметрических*. Такие сущности применяются для обозначения часто обновляемых групп атрибутов или содержания элемента. Другими словами, здесь используется понятие тривиальной текстовой подстановки.

Такие сущности объявляются и вызываются внутри DTD-блока. Естественно, параметрическую сущность сначала необходимо объявить, и только потом — использовать.

Есть и еще одно отличие от обычных сущностей. Для вызова параметрических сущностей перед их псевдонимом мы должны поставить не знак амперсанда, а знак процента. Этот же знак процента ставится и при объявлении параметрической сущности.

Приведем пример. Нам необходимо объявить два разных элемента, причем у каждого из них частично будут совпадать наборы атрибутов. Эту-то совпадающую часть мы и объявим как параметрическую сущность, а затем используем ее в декларировании самих элементов.

```
<!ENTITY % common_atts "
            last_modified CDATA #IMPLIED
            description   CDATA #IMPLIED
            id            ID    #REQUIRED">
<!ELEMENT el_first (#PCDATA)>
<!ELEMENT el_second (#PCDATA)>
<!ATTLIST el_first
            special_attribute CDATA #IMPLIED
            %common_atts;>
<!ATTLIST el_second
            sex (male|female)
            %common_atts;>
```