**Раздел 2.**

**Основы технологии XML**

**2.1 Принципы создания XML-документов. Определение типа (DTD) и схемы (XSD) XML-документа**

**2.2 Преобразование XML-документа средствами XSL**

**2.1. Принципы создания XML-документов. Определение типа (DTD) и схемы (XSD) XML-документа**

1. **История развития XML**

На первых порах Интернет-технологии развивались достаточно хаотично. Для их упорядочения и стандартизации был организован консорциум W3C **(www.w3.org)**, в задачи которого входит формирование политики развития и стандартизации технологий, а также политики формирования *новой платформы* WWW (Web2), в основе которой лежит язык XML и его окружение: группа моделей и стандартов, основанных на нем.

Вместе с исследованиями в области ИИ и представления знаний они формируют новую концепцию Web, называемую Семантический Веб (Semantic Web).  
Язык XML (Extensible Markup Language, расширяемый язык разметки) был разработан рабочей группой XML Working Group консорциума W3C.  
Язык XML - это *метаязык* (язык разметки), являющийся подмножеством стандарта SGML (Standard Generalized Markup Language Стандартного обобщенного языка разметки [ISO 8879]).

Временем рождения XML можно считать 1996 год, в конце которого появился черновой вариант спецификации языка, или 1998, когда эта спецификация была утверждена.

1. **Сравнение языков HTML и XML**

Языки HTML и XML имеют много общего, но разработаны с разными *целями*. Во-первых, оба языка произошли от SGML и являются мета-языками. В обоих в качестве метаданных используются так называемые *теги -* специальные наборы символов, служащие для специфического выделения или пометки некоторых частей документа.

Как и SGML, XML предназначен для *порождения* разнообразных конкретных языков разметки путем определения *конкретных наборов тегов*. Эти определяемые с помощью XML языки являются, таким образом, его конкретизациями.

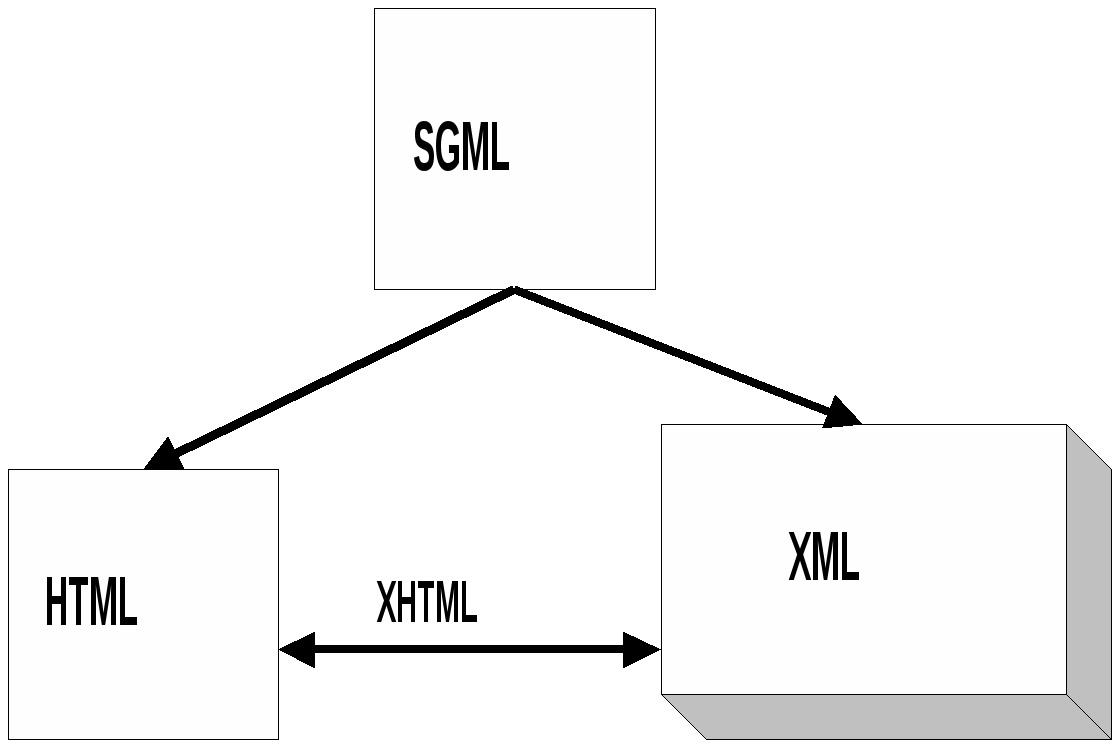
В отличие от XML, язык HTML — *конкретный* (не расширяемый) язык. Функциональность тегов разметки в нем фиксирована.

Для совместимости этих языков был разработан язык XHTML (extensible HTML, расширяемый HTML), являющийся результатом применения правил синтаксиса XML к стандарту HTML.

Последняя версия спецификации XML находиться по адресу [http://www.w3c.org/TR/rec-xml](http://www.w3c.org/tr/rec-xml).

Не заменяя HTML, XML в настоящее время используется в сочетании с ним, расширяя возможности Web-страниц для:

* виртуального представления документов любого типа;
* сортировки, фильтрации, упорядочения, поиска и манипулирования информацией иными способами;
* представления информации в структурированном виде.



Как заявляют сами разработчики, XML был создан для взаимодействия с HTML и совместного с ним использования.

1. **Цели и задачи XML**

При разработке языка XML ставились следующие задачи:

1. XML должен быть пригоден для непосредственного использования в Интернет.
2. XML должен иметь широкий круг применения.

*Хотя основным его назначением является распространение информации в Web через серверы и программы-браузеры, XML также разработан для использования его другими программами. Например, XML применяется для обмена информацией между финансовыми программами, для распространения и обновления программных продуктов, а также написания голосовых сценариев при доставке информации по телефону.*

1. XML должен быть совместим с SGML.

*XML является специализированной ветвью SGML. Преимущество здесь заключается в простоте адаптации программных средств SGML для работы с HTML.*

1. Обработчики документов XML должны быть просты в написании.

*Для практического использования XML необходимо, чтобы было достаточно просто писать браузеры и другие программы, обрабатывающие XML-документы. На деле основной причиной выделения XML из SGML была доступность написания программ для обработки XML-документов.*

1. Количество необязательных свойств в XML должно быть сведено к минимуму.

*Минимальное число дополнительных функций в XML упрощает написание программ для обработки XML-документов. Изобилие дополнительных подключаемых функций в SGML стало основной причиной, обусловившей его практическую непригодность для представления Web-документов. Дополнительные функции SGML требуют переопределения символов-разделителей для тегов (обычно ) и пропуск конечного тега с целью обнаружения процессором конца элемента. При строгом написании программы обработки SGML-документов необходимо учитывать возможность появления всех дополнительных функций, даже если они редко встречаются*.

1. XML документы должны быть удобны для чтения и достаточно понятны.

*XML призван стать* ***lingua franca (универсальным языком)*** *для обмена информацией среди пользователей и программ по всему миру. В соответствии с этой концепцией пользователи, а также специализированные программы, должны иметь возможность создавать и прочитывать XML-документы. Доступность и прозрачность для пользователя выделяют XML из большинства других форматов, применяемых при построении баз данных и текстовых документов.  
Пользователь может легко прочесть XML-документ, поскольку он описан простым текстом и имеет логичную иерархическую структуру в виде дерева. Вы можете упростить XML-документы, назначив информативные имена для элементов, атрибутов и объектов, а также добавив полезные комментарии.*

1. Подготовка XML документа должна осуществляться быстро.

*XML станет общепринятым стандартом лишь в том случае, если программисты и пользователи примут его. Следует создать этот стандарт до того, как общество примет альтернативные стандарты, которые все быстрее создаются компаниями-разработчиками программного обеспечения.*

1. Процедура построения XML документа должна быть формальной и точной.

*Спецификация XML написана на формальном языке, используемом для представления компьютерных языков, с нотацией, известной как расширенная форма Бэкуса-Наура (Extended Backus-Naur Form (EBNF)). Этот формальный язык, хотя и достаточно сложен для восприятия, лишен двусмысленности и существенно облегчает написание XML-документов, а в особенности программ для их обработки.*

1. Процедура создания XML документов должна быть проста.

*При практическом использовании XML как языка разметки для Web-документов упрощается не только написание обрабатывающих программ, но и процесс создания самих XML-документов.*

1. Краткость при разметке XML документа имеет минимальное значение.

*В соответствии с пунктом 6 (XML-документ должен быть ясным и понятным для пользователя), XML-разметка не должна быть излишне сжатой, чтобы не вступать в противоречие с указанной целью.*

## **4. Определение и структура XML-документа**

Любой документ можно представить в виде некого набора символов, разделенных на две группы: первая группа определяет содержимое документа, вторая – специальные наборы символов (теги), служащие для специфического выделения или пометки некоторых частей первой группы. В XML-документах первая группа носит название "текст" или "контент", а вторая, представляющая собой систему обозначений, которые поддаются интерпретации, называется разметкой. В языке HTML, к примеру, к разметке относятся все *теги*, без исключения.

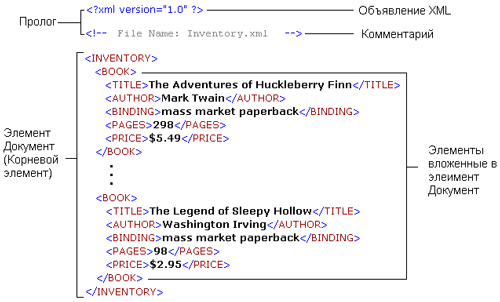
Разметка XML-документа представляет собой несколько *более сложное понятие*, чем в языке HTML. Это связано с тремя следующими обстоятельствами:  
*1. XML-документ может содержать инструкции обработки самого себя.* Это означает, что обработчик XML-документа (процессор) является посредником между этим документом и некоторым приложением или приложениями. Такими приложениями могут быть базы данных, сервер каталога, язык программирования, работающий на стороне сервера, и т. п.  
  
*2. Синтаксис языка XML определен весьма жестко и не допускает многозначного толкования своих правил.* Причем любое отклонение от этих правил влечет за собой отказ процессора обрабатывать весь документ. Поэтому в языке XML количество ограничений на синтаксис документа, о которых необходимо помнить разработчику, гораздо больше, чем в языке HTML.  
  
*3. В отличие от языка HTML, разработчик XML-документов сам определяет названия тегов и, может быть, правила их интерпретации.* Если такие правила присутствуют, то они отделены от непосредственно XML-кода и образуют так называемый сценарий XSLT (XSL Transformation). Иными словами, если XML-файл служит не только для хранения каких-либо данных, но и для их представления, (например, в браузере), то способы представления этих данных вынесены в отдельный XSL-файл (XSL – расширяемый язык стилей).

Структура информации, содержащейся в XML-файле, представляет собой иерархическую модель данных (как и HTML-документа), то есть, вся символьная информация: разметка или контент представляет собой совокупность узлов некоторого дерева. По правилам XML в корректном документе должен присутствовать единственный корневой элемент. Все остальные элементы являются его непосредственными потомками или потомками других его потомков. Процесс обработки такого документа начинается с первого или корневого элемента, затем обрабатываются узлы дерева второго уровня, затем — третьего и т. д. Подавляющее большинство ошибок возникает в результате того, что наборы символов размечены не однозначно, т. е. процессор не может определить, к какому именно узлу следует отнести данный набор.

## **5. Создание XML-документа**

XML-документ представляет собой *обычный текстовый файл*, в котором при помощи специальных тегов создаются элементы данных, последовательность и вложенность которых определяет *структуру документа и его содержание*.

### Структура XML-документа

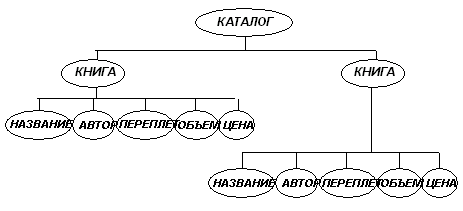
Как и HTML-документ, XML-документ состоит из двух частей: *Заголовка (пролога)* (как тег HEAD в HTML) и элемента Документ (его также называют корневым элементом).  
  
  
**Заголовок XML-документа**

Заголовок (пролог) согласно спецификации языка XML, подчиняется следующим правилам синтаксиса:  
1) заголовок должен начинаться с символов   
2) перед начальными символами заголовка не должно быть других символов;  
3) заголовок должен заканчиваться символами ?> ;  
4) после начальных символов должно стоять слово xml;  
5) указание версии с помощью конструкции version =" . . . " обязательно;  
6) номер версии на данный момент *— 1.0;*  
7) номер версии должен быть заключен в кавычки.

Помимо номера версии заголовок XML- документа может включать в *объявление* кодировки документа, которая определяется с помощью конструкции encoding =". . . ".

**Элемент Документ**  
Второй основной частью XML-документа является единый элемент *Документ*, или корневой элемент, который в свою очередь содержит дополнительные элементы.  
***Примечание.*** *Элемент Документ в XML-документе похож на элемент BODY на HTML-странице, за исключением того, что вы можете присвоить ему любое допустимое имя.*  
**Важно!**  
1. Язык XML является чувствительным к регистру символов, поэтому как открывающий, так и закрывающий теги должны быть записаны символами в одном регистре.  
tag и Tag – это разные теги в отличие от HTML!  
2. Не допускается один или несколько пробелов перед открывающей скобкой тега, хотя в любом другом месте пробелы допустимы, То же касается и символов концов строк.   
3. Применение закрывающего тега всегда обязательно!

Имена *элементов* в XML-документе (такие как INVENTORY, BOOK и TITLE в приведенном выше примере) не являются определениями языка XML. Вы всего лишь назначаете эти имена при создании определенного документа. Для ваших элементов вы можете выбирать любые корректно заданные имена (LIST вместо INVENTORY, либо ITEM вместо BOOK).  
  
***Примечание****. Хотя нет спец. запрещений, лучше имена элементов писать латиницей.*

В предыдущем примере XML-документ имеет иерархическую структуру в виде дерева с элементами, вложенными в другие элементы, и с одним элементом верхнего уровня (в нашем примере – INVENTORY) – он носит название элемент Документ или корневой элемент, – который содержит все другие элементы. Структуру описанного в примере документа можно представить, как показано на рисунке.  
  
  
  
Таким образом, с помощью XML вы можете описать иерархическую структуру документа, такого как книга, содержащего части, главы и разделы.  
  
**Способы отображения XML-документа**

Есть *три* основных способа сообщить браузеру как обрабатывать и отображать каждый из созданных XML-элементов.   
  
*Таблица стилей*. С помощью данного метода вы связываете таблицу стилей с XML-документом. Таблица стилей представляет собой отдельный файл, содержащий инструкции для форматирования индивидуальных XML-элементов. Вы можете использовать либо каскадную таблицу стилей (Cascading Style Sheet - CSS), которая также применяется для HTML-страниц, либо расширяемую таблицу в формате языка стилевых таблиц (Extensible Stylesheet Language - XSL), обладающую более широкими возможностями, нежели CSS, и разработанную специально для XML-документов.

* *Связывание данных.* Этот метод требует создания HTML-страницы, связывания с ней XML-документа и установления взаимодействий стандартных HTML-элементов на странице, таких как SPAN или TABLE, с элементами XML. В дальнейшем HTML-элементы автоматически отображают информацию из связанных с ними XML-элементов.
* *Написание сценария.* В этом методе вы создаете HTML-страницу, связываете ее с XML-документом и имеете доступ к отдельным XML-элементам с помощью специально написанного кода сценария (JavaScript или VBScript). Браузер воспринимает XML-документ как объектную модель документа (Document Object Model - DOM), состоящую из большого набора объектов, свойств и команд. Написанный код позволяет осуществлять доступ, отображение и манипулирование XML-элементами.

Вы можете открыть XML-документ непосредственно через Internet Explorer, точно так же, как вы бы открыли HTML Web-страницу.  
  
Если XML-документ не содержит связи с таблицей стилей, Internet Explorer помечает различные составные части документа различным цветом, чтобы облегчить их распознавание, а также представляет элемент Документ в виде иерархического дерева с возможностью свертывания и развертывания структуры и просмотра с меньшей или большей степенью детализации.

Если же XML-документ имеет связь с таблицей стилей, Internet Explorer отобразит только символьные данные из элементов документа, отформатировав их в соответствии с правилами, установленными в таблице стилей.

### Определение корректного документа

*Корректно сформированным (правильно оформленным, well-formed)* называется документ, отвечающий минимальному набору критериев соответствия для ХМL-документа. Когда вы создаете корректно сформированный XML-документ, вы можете добавлять элементы и вводить данные непосредственно в ваш документ, как вы это делаете при создании HTML-документов.   
  
Имена, которые вы присваиваете элементу (тегу), называются *типом элемента.*  
  
**Правила создания корректного документа:**

**Документ должен иметь только один элемент верхнего уровня (элемент Документ или корневой элемент).** Все другие элементы должны быть вложены в элемент верхнего уровня.

**Элементы должны быть вложены упорядоченным образом.** То есть, если элемент начинается внутри другого элемента, он должен и заканчиваться внутри этого элемента.

**Каждый элемент должен иметь начальный и конечный тег.** В отличие от HTML, в XML не разрешается опускать конечный тег – даже в том случае, когда браузер в состоянии определить, где заканчивается элемент (за исключением пустых элементов – т.е. элементов, не имеющих содержимого)

**Имя типа элемента в начальном теге должно в точности соответствовать имени в соответствующем конечном теге.**

**Имена типов элементов чувствительны к регистру, в котором они набраны.** В действительности весь текст внутри XML-разметки является чувствительным к регистру.

### Составные части корректно сформированного ХМL-документа

Как мы уже знаем, XML-документ состоит из двух основных частей: пролога и элемента Документ (корневого элемента). Помимо этого, корректно сформированный XML-документ может содержать *комментарии*, *инструкции по обработке*, *пробелы*. На следующем рисунке приведен пример корректно сформированного XML-документа, отражающий различные части документа и включения, которые вы можете добавлять в каждую из частей.

  
***Примечание.*** *XML-объявление в этом примере также включает в себя объявление документа автономным (****standalone='yes'****). Это объявление может использоваться в некоторых XML-документах с целью упростить обработку документа.*

В рассматриваемом примере имеется комментарий в прологе, а также другой комментарий, следующий за элементом Документ.

**Написание определений DTD: общие принципы.**

**Ассоциирование DTD с документом XML**

Для связывания декларации DTD с экземпляром документа в версии XML 1.0 предлагается специальная декларация DOCTYPE. Она должна следовать после декларации XML и предшествовать любым элементам документа. Тем не менее, между декларациями XML и DOCTYPE могут находиться комментарии и команды обработки.

Декларация DOCTYPE содержит ключевое слово DOCTYPE, за которым следует имя корневого элемента документа, а затем конструкция с декларациями содержания. Перед разъяснением этого утверждения рассмотрим пример расположения декларации DOCTYPE в экземпляре документа. Ниже приводятся первые три строчки документа XML:

.. <xml version ="1.0" ?>

<!DOCTYPE root\_element\_name … >

<root\_element\_name > …

Можно написать внешнее подмножество деклараций в отдельном файле DTD, включить внутреннее подмножество в тело декларации DOCTYPE или сделать то и другое. В последнем случае (смешение внутренних и внешнихDTD) во внутренних DTD могут быть заданы новые декларации или переписаны те, что содержатся во внешних (по определению спецификации XML анализаторы сначала читают внутреннее подмножество, и потому содержащиеся там декларации пользуются приоритетом).

Декларации XML могут содержать атрибут standalone, принимающий только значения "yes" и "nо". Если значение атрибута равно yes, то внешние для экземпляра документа декларации не влияют на информацию, передаваемую документом использующему его приложению. Значение no показывает, что существуют внешние декларации со значениями, необходимыми для правильного описания содержания документа - например конкретные значения по умолчанию. На практике необязательный атрибут standalone используется редко. Наличие этого атрибута со значением, yes не гарантирует отсутствия внешних зависимостей любого типа. Просто внешние зависимости в этом случае не приведут к ошибке в документе, если не будут включены в обработку. Таким образом, в основном этот атрибут представляет собой знак для анализаторов и других приложений, показывающий, нужно ли им использовать какое-либо внешнее содержание.

Блок внутренней декларации разметки тега DOCTYPE состоит из левой квадратной скобки, списка деклараций и правой квадратной скобки:

<! DOCTYPE root\_element\_name […здесь находятся декларации

внутреннего подмножества ... ]>

Внутренние DTD очень полезны, они всегда содержатся в использующем их документе и поэтому их получение анализатором не представляет проблем. Однако внутренние DTD значительно увеличивают размер документа. На практике внутренние DTD чаще всего применяются одновременно с внешними для добавления новых декларации. Если там объявлен какой-либо объект, продекларированный также и во внешнем определении DTD, приоритетом пользуется внутреннее. Это позволяет осуществлять тонкую настройку деклараций для нужд конкретных документов.

Внешние DTD в некоторых отношениях более гибкие. В данном случае декларация DOCTYPE состоит из обычного ключевого слова и имени корневого элемента, за которым следует еще одно ключевое слово SYSTEM либоPUBLIC, обозначающее источник внешнего определения DTD, а за ним - локализация этого определения. Если ключевое слово SYSTEM, DTD обязано непосредственно и явным образом находиться по указанному URL адресу.

Если внешние DTD переписываются очень часто, они начинают терять свое значение, а это признак плохого первоначального проекта.

Ключевое слова PUBLIC предназначено для хорошо известных словарей. Приложение, анализирующее документ из этого словаря, должно использовать некоторую стратегию по поиску соответствующего DTD.

Стандарт XML 1.0 допускает у декларации PUBLIC наличие как публичного URI, так и системного идентификатора. Если работающее с документом приложение или анализатор не могут найти DTD по идентификатору URI с ключевым словом PUBLIC, оно должно использовать системный идентификатор.

**Основные декларации разметки**

Допустимое в документе XML содержание определяется с помощью четырех типов декларации разметки в DTD. В следующей далее таблице показаны связанные с этими декларациями ключевые слова и их значения:

|  |  |
| --- | --- |
| **Конструкция DTD** | **Значение** |
| ELEMENT | Декларация типа элемента XML |
| ATTLIST | Декларация атрибутов, которые могут быть назначены конкретным типам элементов, а также разрешенных значений этих атрибутов |
| ENTITY | Декларация повторно используемого содержания |
| NOTATION | Декларация форматирования для внешнего содержания, которое не должно быть проанализировано (например, двоичные данные), а также для внешних приложений, обрабатывающих содержание |

Первые два типа связаны с информацией, которую мы рассчитываем найти в документе XML, - элементами и атрибутами.

Последние два типа используются для поддержки. Особенно облегчают жизнь разработчика словаря XML сущности. Как правило, они состоят из содержания, которое настолько часто используется в DTD или документе, что оправдывает создание специальной декларации. Применение этой декларации напоминает оператор include в языках C/C++, когда в качестве замены для содержания используется имя.

Нотации описывают содержание, разработанное не на языке XML. Используются они для того, чтобы объявить конкретный класс данных и связать его с внешней программой. Эта внешняя программа становится обработчиком объявленного класса данных. Например, связав с документом изображение в формате JPEG, разработчик желает, чтобы программа приняла и визуализировала двоичные данные в этом формате. Конечно, в таком случае документ зависит от того, какой обработчик имеется в системе, получающей документ. В интересах портативности и переносимости некоторые авторы не приводят ссылки на обработчики. В таком случае нотация превращается просто в механизм набора текста.

**Объявление элементов.**

Каждый элемент документа XML должен быть описан. Объявление элемента начинается с символов <!ELEMENT, после которых через пробел идет имя элемента и его содержимое. Заканчивается объявление символом "больше". По своему содержанию элементы делятся на четыре группы.

1. Пустой элемент - может иметь атрибуты, но не содержит текст или порожденные элементы. Объявляется следующим образом: после имени элемента указывается ключевое слово EMPTY.
2. Пример:

<!ELEMENT element\_name EMPTY>

1. Элемент содержит только порожденные элементы, но не текст. Объявляется следующим образом: после имени элемента в скобках через запятую перечисляются все вложенные элементы. Причем вложенные элементы должны следовать в XML документе в том порядке, в каком они перечислены в объявлении.
2. Пример:

<ELEMENT element\_name ( elem\_1,elem\_2)>

1. Элемент содержит не только порожденные элементы, но и текст. Объявляется следующим образом: после имени элемента в скобках указывается ключевое слово #PCDATA, после которого через запятую, как и в предыдущем случае, перечисляются все вложенные элементы (если они имеются).
2. Пример:
3. <ELEMENT element\_name (#PCDATA, elem\_1,elem\_2)>

<ELEMENT element\_name (#PCDATA)>

1. Элемент, открытый для любого содержания. Объявляется следующим образом: после имени элемента указывается ключевое слово ANY.
2. Пример:

<ELEMENT element\_name ANY>

Иногда из нескольких вложенных элементов или списков (список элементов указанных в скобках) может встретиться только один. В таком случае их имена перечисляются через вертикальную черту( | ). Например:

<!ELEMENT element\_name (elem\_1,(elem\_2|elem\_3))> - элемент element\_name

должен содержать элемент elem\_1,

а затем либо elem\_2, либо elem\_3.

Элементы появляются именно в таком порядке.

Если вложенный элемент можно записать в объявляемом элементе несколько раз, то необходимо это указать используя звездочку, плюс или вопросительным знак.

? - элемент или список может встретиться нуль или один раз;

\* - элемент или список может встретиться нуль или несколько раз;

+ - элемент или список может встретиться один или несколько раз.

**Объявление атрибутов.**

Атрибуты элемента объявляются после объявления самого элемента. Все атрибуты одного элемента объявляются сразу, одним списком. Список начинается с символов <!ATTLIST, после них через пробел следует имя элемента, к которому относятся атрибуты. Затем идет имя атрибута, его тип или список значений, которые он может принимать (все значения перечисляются через вертикальную черту, в скобках), признак обязательности присутствия атрибута в элементе или значение по умолчанию (это значение будет использовано, если атрибут не записан явно в XML документе).

Тип атрибута записывается одним из ключевых слов:

* CDATA - строка символов.
* ID - уникальный идентификатор, однозначно определяющий элемент, в котором встретился этот атрибут; значения такого атрибута не должны повторяться в документе. Они играют ту же роль, что и первичные ключи в таблицах баз данных.
* IDREF - идентификатор, содержащий одно из значений атрибутов типа id, используется в качестве ссылки на другие элементы.
* IDREFS - идентификатор, содержащий набор значений атрибутов типа id, перечисленных через пробелы; тоже используется в качестве ссылки сразу на несколько элементов.
* ENTITY - имя непроверяемой анализатором сущности объявленных в этом же описании DTD.
* ENTITIES - имена непроверяемых сущностей.
* NMTOKEN - слово, содержащее только символы, применяемые в именах. Атрибуты этого типа могут содержать имена других элементов или атрибутов, например, для того чтобы ссылаться на них.
* NMTOKENS - слова, перечисленные через пробелы.
* NOTATION - обозначение расшифрованное в описании DTD.

Признак обязательности записывается с использование ключевых слов:

1. #REQUIRED - атрибут надо обязательно записывать в элементе;
2. #IMPLIED - атрибут необязателен, у него нет значения по умолчанию;
3. #FIXED - у атрибута есть только одно значение, которое записывается тут же, через пробел.

Пример:

<!ATTLIST city type (город | поселок | деревня) "город">

<!ATTLIST pre xml:lang NMTOKEN "ru\_RU">

<!ATTLIST pre xml: space (default | preserve) "preserve">

**Объявление сущности.**

Ссылки на сущности используются как краткие обозначения для громоздких или часто повторяющихся фрагментов документа XML. Сами сущности подставляемые в документ вместо ссылок, объявляются в описании DTD.

Все сущности можно разделить на три группы:

1. внутренние сущности - задаются при объявлении сущности. Объявление начинается с символов <!ENTITY, после которых через пробел записывается имя сущности и ее значение в кавычках. Например:

<!ENTITY lang "XML">

После такого объявления программа-анализатор, увидев в документе ссылку на сущность &lang;, заменить ее на строку XML. Ссылку на сущность можно применять тут же, в описании DTD, уже в следующем объявлении.

1. внешние сущности - содержатся в отдельных файлах или встроены в программу-анализатор. Для них указывается одно из слов SYSTEM или PUBLIC после которого записывается место их расположения. После ключевого слова SYSTEM указывается URI адрес. После слова PUBLIC идет какое-то общеизвестное объявление, после которого через пробел также указывается URI адрес, которым программа-анализатор воспользуется, если не поймет указанного объявления.
2. параметризованные сущности - используются только внутри описания DTD. Объявление начинается с символов <!ENTITY, после которых через пробел записывается знак процента (%), имя сущности и ее значение в кавычках. Например:

<!ENTITY % lang "ru\_RU">

Ссылка на параметризованную сущность начинается не с амперсанда, а со знака процента, в примере- %lang;. Введение этой ссылки удобно тем, что при смене языка надо будет поменять значение ru \_Ru только в одном месте описания.

**Объявление инструкций по обработке.**

Объявление инструкций по обработке начинается с символов <!NOTATION, после них записывается имя инструкции, одно из ключевых слов SYSTEM или PUBLIC (причем слова SYSTEM и PUBLIC здесь равнозначны), затем в кавычках - ее расшифровка.

Пример.

<!NOTATION image-gif SYSTEM "viewer.ехе">

Это объявление связывает обозначение image-gif с программой обработки изображений, находящейся в файле viewer.exe.

**Недостатки и особенности определений DTD.**

Описание структуры документа XML, выполненное средствами DTD имеет ряд недостатков и ограничений:

1. Синтаксис DTD отличается от синтаксиса XML. XML документ не может обработать проверяющий на допустимость анализатор XML, но он не обязан предоставлять связанный с этим документом DTD вызвавшему его приложению. К тому же, если мы хотим из приложения обрабатывать определения DTD, нам потребуется отдельный механизм анализа.
2. Определения DTD представляют собой закрытые конструкции. Вся информация должна быть включена только в одно определение DTD. Это не кажется ограничением, пока нет необходимости позаимствовать декларацию или какую-либо иную полезную конструкцию из другого определения DTD.
3. Сегментация недопустима, если не считать ситуаций, когда определение DTD можно логически сконструировать таким образом, чтобы оно включало вложенные определения DTD.
4. В DTD практически отсутствует информация о типах данных, точном количестве повторений вложенного элемента и некоторые другие необходимые подробности.

***Использование схем XML для описания структуры документов XML***

XML-схемы являются альтернативой DTD. Они, также как и DTD, определяют набор используемых элементов, идентифицируют элементы, которые могут использоваться внутри других элементов, определяют возможные атрибуты для каждого элемента. По сравнению с DTD, схемы обеспечивают более понятный способ описания документов. Схемы XML, в отличие от DTD, являются XML – документами.

Как отмечалось ранее, в DTD для описания содержимого элемента используется способ, похожий на описание цепочек символов, допускаемых автоматом. В схемах XML используется способ, более привычный для программиста: определяются типы данных и указывается принадлежность элементов XML к этому типу данных.

**Пример XML-схемы**

**Файл XML:**

<?xml version="1.0" encoding="Windows-1251"?>

<languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="Example\_schema.xsd">

<language id="1">

<name>HTML</name>

<year>01.01.1990</year> 16

<howold>19</howold>

</language>

<language id="2">

<name>XML</name>

<year>01.01.1998</year>

<howold>11</howold>

</language>

<language id="3">

<name>SGML</name>

<year>01.01.1986</year>

<howold>23</howold>

</language>

<empty attr1="1" attr2="текст"/>

<CDATA\_Example>

<![CDATA[<<<<<<<<<< >>>>>>>>>]]>

</CDATA\_Example>

</languages>

**Файл XSD:**

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

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:complexType name="languageType">

<xsd:sequence>

<xsd:element name="name" type="xsd:string"/>

<xsd:element name="year" type="xsd:string"/>

<xsd:element name="howold" type="xsd:integer"/>

</xsd:sequence>

<xsd:attribute name="id" use="required" type="xsd:string"/>

</xsd:complexType>

<xsd:complexType name="emptyType">

<xsd:attribute name="attr1" type="xsd:string" use="required"/>

<xsd:attribute name="attr2" type="xsd:string" use="required"/>

</xsd:complexType> 17

<xsd:element name="languages">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="language" type="languageType" maxOccurs="unbounded"/>

<xsd:element name="empty" type="emptyType"/>

<xsd:element name="CDATA\_Example" type="xsd:string"/>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:schema>

Связать XML документ с его XSD схемой можно разными способами:

1. подать файлы со схемой на вход анализатора.
2. задать файлы со схемой как свойство анализатора.
3. указать прямо в документе XML.

Рассмотрим 3 способ более подробно.

Если элементы документа не принадлежат никакому пространству имен и записаны без префикса, то в корневом элементе документа записывается атрибут noNamespaceSchemaLocation, указывающий расположение файла со схемой в форме URI:

< root\_element\_name xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance

xsi:noNamespaceSchemaLocation="file\_name.xsd">

В этом случае в схеме не должно быть целевого пространства имен.

Если же элементы документа относятся к некоторому пространству имен, то применяется атрибут schemaLocation, в котором через пробел перечисляются пространство имен и расположение файла со схемой, описывающей это пространство имен.

Элементы, из которых будет состоять документ XML, объявляются в схеме компонентом element:

<xsd:element\_name="имя элемента" tуре="тип элемента"

minOccurs="Haменьшее число появлений элемента в документе"

maxOccurs="наибольшее число появлений" />

Значение по умолчанию необязательных атрибутов minOccurs и maxOccurs равно 1. Это означает, что если эти атрибуты отсутствуют, то элемент должен появиться в документе XML ровно один раз.

Указание типа элемента в атрибуте type удобно, если это встроенный простой тип или тип, определенный заранее. Тогда в атрибуте type можно записать только имя типа.

Если же тип элемента определяется здесь же, то определение типа элемента лучше вынести:

<xsd:element name="имя элемента">

Определение типа элемента

</xsd:element>

Объявление атрибута элемента тоже несложно:

<xsd:attribute name=" имя атрибута" type="тип атрибута"

use="обязательность атрибута"

default="значение по умолчанию" />

Необязательный атрибут use принимает три значения:

1. optional - описываемый атрибут необязателен (это значение по умолчанию);
2. required - описываемый атрибут обязателен;
3. prohibited - описываемый атрибут неприменим. Это значение полезно при определении подтипа, чтобы отменить некоторые атрибуты базового типа.

Определение типа атрибута, - а это должен быть простой тип, - можно вынести в содержимое элемента attribute:

<xsd:attribute name="имя атрибута">

Тип атрибута

</xsd:attribute>

**Определение простых типов**

Простой тип в схемах XML определяется компонентом схемы simpleType, имеющим вид

<xsd:simpleType name="имя типа" >Определение типа</xsd:simpleType">

**Определение новых типов простых элементов**

Кроме встроенных типов в схемах XML можно определить новые типы простых элементов. Они вводятся как

1. сужение (restriction) встроенного или ранее определенного простого типа,
2. список (list) простых типов
3. объединение (union) простых типов.

Сужение простого типа определяется компонентом restriction, в котором атрибут base указывает сужаемый простой тип, а в содержимом задаются ограничения, выделяющие определяемый простой тип.

Теги задающие ограничения, называются фасетками (facets). Вот их список:

1. <maxExclusive> - наибольшее значение, которое уже не входит в определяемый тип;
2. <maxInclusive> - наибольшей значение определяемого типа;
3. <minExclusive> - наименьшее значение, уже не входящее в определяемый тип;
4. <minInclusive> - наименьшее значение определяемого типа;
5. <totalDigits> - общее количество цифр в определяемом числовом типе - сужении типа decimal;
6. <fractionDigits> - количество цифр в дробной части числа;
7. <length> - длина значений определяемого типа;
8. <maxLength> - наибольшая длина значений определяемого типа;
9. <minLength> - наименьшая длина значений определяемого типа;
10. <enumeration> - одно из перечислимых значений;
11. <pattern> - регулярное выражение;
12. <whitespace> - применяется при сужении типа string и определяет способ преобразования пробельных символов <'>\n<'>, <'>\r<'>, <'>\t. Атрибут value этого тега принимает одно из трех значений:
    * preserve - не убирать пробельные символы;
    * replace - заменить пробельные символы пробелами;
    * collapse - после замены пробельных символов пробелами убрать начальные и конечные пробелы, а из нескольких подряд идущих пробелов оставить только один.

В тегах-фасетках можно записывать следующие атрибуты, называемые базисными фасетками (fundamental facets):

1. ordered- задает упорядоченность определяемого типа, принимает одно из трех значений:
   * false - тип неупорядочен;
   * partial - тип частично упорядочен;
   * total - тип полностью упорядочен;
2. bounded - задает ограниченность или неограниченность типа значением true или false;
3. cardinality - задает конечность или бесконечность типа значением finite или countably infinite;
4. numeric - показывает, числовой этот тип или нет, значением true или false.

Список определяется компонентом list, в котором атрибутом itemType указывается тип элементов определяемого списка. Тип элементов списка можно определить и в содержимом элемента list.

При определении списка можно применять фасетки <length>, <minLength>, <maxLength>, <enumeration>, <pattern>.

Простой тип-объединение определяется компонентом union, в котором атрибутом memberTypes можно указать имена объединяемых типов. Например:

<xsd: union memberTypes="xsd: string xsd; integer lisstOfInteger" />

Другой способ - записать в содержимом компонента union определения простых типов, входящих в объединение.

**Определение сложных типов**

Тип элемента называется сложным, если в элемент вложены другие элементы и/или в открывающем теге элемента есть атрибуты.

Сложный тип определяется компонентом complexType, имеющим вид:

<xsd:complexType name="имя типа" >

Определение типа

</xsd:complexType>

Необязательный атрибут name задает имя типа, а в содержимом компонента complexType описываются элементы, входящие в сложный тип, и/или атрибуты открывающего тега.

Определение сложного типа можно разделить на три группы:

1. определение типа пустого элемента (элемента, не содержащего тела, а содержащего только атрибуты в открывающем теге).

Каждый атрибут объявляется одним компонентом attribute, например:

<xsd:complexType name="imageType">

<xsd:attribute name="href" type="xsd:anyURI" />

</xsd:complexType>

1. определение типа элемента с простым телом (элемента, содержащего тело простого типа и атрибуты в открывающем теге).

Этот тип отличается от простого типа только наличием атрибутов и определяется компонентом simpleContent. В теле этого компонента должен быть либо компонент restriction, либо компонент extension. Атрибутомbase задают тип (простой) тела описываемого элемента.

В компоненте extension указываются атрибуты открывающего тега описываемого элемента.

В компоненте restriction, кроме атрибутов, описывается простой тип содержимого элемента и/или фасетки, ограничивающие тип, заданный атрибутом base.

1. определение типа элемента, содержащего вложенные элементы.

Если значениями определяемого сложного типа будут элементы, содержащие вложенные элементы, то перед тем, как перечислять их описания, надо выбрать модель группы (model group) вложенных элементов. Дело в том, что вложенные элементы, составляющие определяемый тип, могут появляться или в определенном порядке, или в произвольном порядке, кроме того, можно выбирать только один из перечисленных элементов. Эта возможность и называется моделью группы элементов. Она определяется одним из трех компонентов: sequence, all или choice.

Компонент sequence применяется в том случае, когда перечисляемые элементы должны записываться в документе в определенном порядке.

Если же вместо компонента xsd:sequence записать компонент xsd:all, то элементы можно перечислять в любом порядке.

Компонент choice применяется в том случае, когда надо выбрать один из нескольких элементов.

1. Определение типа элемента со сложным телом

При определении сложного типа можно воспользоваться уже определенным, базовым, сложным типом, расширив его дополнительными элементами, или, наоборот, удалив из него некоторые элементы. Для этого необходимо применить компонент complexContent. В этом компоненте, так же как и в компоненте simpleContent, записывается либо компонент extension, если надо расширить базовый тип, либо компонентrestriction, если нужно его сузить. Базовый тип указывается атрибутом base, так же как и при записи компонента simpleContent, но теперь это должен быть сложный, а не простои тип. При сужении базового типа компонентом restriction надо перечислить те элементы, которые останутся после сужения.

### 2.1 Преобразование XML-документа средствами XSL

#### Средства форматирования документов XML

Основными типами выходных документом при преобразованиях XSLT являются документы XML, текстовые документы и документы HTML. Конечным результатом преобразования является представление выходного документа в оформлении, которое зависит как от содержания документа, так и носителя, на который выводится документ (экрана дисплея, печатающего устройства и т.д.). Получение нужной структуры вывода документа (разметки страниц, параметров шрифтов и т.п.) называется форматированием документа. Для текстового документа средства форматирования крайне ограничены (фактически это только вставка в выводимый документ символов пробела и символов переноса строки). Для форматирования документа HTML можно использовать как элементы и атрибуты форматирования, определенные в спецификации HTML, так и средства каскадных таблиц стилей (CSS). Средства языка XML описывают только структуру документа, но не содержат никаких инструментов форматирования. Для форматирования документа XML можно использовать либо рассмотренное выше ассоциирование таблиц CSS с документами XML, либо выполнять форматирование документа с использованием средств языка Java (DOM-анализаторов). Первый способ предоставляет очень ограниченные средства форматирования (только для содержимого элементов документа XML), второй способ требует написания программ, которые к тому же можно использовать только для форматирования того документа, для которого эта программа написана.

Поэтому наиболее целесообразно использовать для форматирования документов специально разработанный консорциумом W3 расширенный язык таблиц стилей – XSL (eXtensible Stylesheet Language). Первая спецификация этого языка (версия 1.0) вышла в ноябре 2001 года. Следующая версия языка – XSL 1.1 утверждена консорциумом W3 в декабре 2006 года. Далее рассматривается именно эта версия. Однако следует иметь в виду, что пока не все возможности этой версии реализованы в программных средствах, реализующих язык XSL.

В настоящее время начата разработка новой версии XSL – XSL 2.0.

#### Этапы форматирования документа XML

При форматировании документа XML в качестве начального этапа может быть выполнено преобразование с использованием языка XSLT (рис. [5.3.2](file:///E:\КОЛЛЕДЖ\УМК_ПСС\Лекции\Lect4-08.doc#fig_5_3_2)), в результате которого преобразованный документ представляется в виде иерархической (древовидной) структуры или просто дерева.

Это дерево или дерево исходного документа (если этап преобразования отсутствует) рассматривается программой форматирования документа XML – процессором XSL как дерево элементов и атрибутов XML в пространстве имён "fo".

Процесс форматирования состоит из трех этапов.

На первом этапе форматирования дерево элементов и атрибутов XML преобразуется в дерево объектов форматирования (рис.).

Дерево преобразования или исходного документа

Дерево объектов

форматирования и свойств

Объективизация

Элемент

**fo:block**

Атрибуты

**start-indent="2em"**

**font-size="20pt"**

Объективизация

Объект форматирования

**fo:block**

Свойства

**start-indent="2em"**

**font-size="20pt"**

а)

б)

Рис. Первый этап форматирования: а) всего документа;

б) отдельного элемента и атрибута

Объект форматирования – FO (Formatting Object) содержит сведения о разбивке на страницы, компоновке и стиле, которые будут применяться к содержимому этого объекта форматирования. С каждым объектом форматирования связан определенный набор свойств, которые управляют форматированием этого объекта. В целом словарь объектов форматирования, поддерживаемых XSL (набор элементов в пространстве имен fo:) представляет собой набор типографских абстракций, необходимых для стилевого оформления документа XML.

В процессе выполнения первого этапа, называемого «объективизацией», узлы элементов исходного дерева преобразуются в узлы объектов форматирования, а узлы атрибутов – в спецификации свойств.

Как часть этапа объективизации, символы результирующего дерева замещаются узлами элемента fo:character, определяющего отдельный символ. Для вставляемых в результирующее дерево внешних объектов (элемент fo:instream-foreign-object) и элементов-потомков объявлений (элемент fo:declarations) в пространствах имён, не являющихся именами XSL, «объективизация не выполняется. Вместо этого объект, представляющий один из приведенных выше элементов, указывает на подходящий узел в дереве элементов и атрибутов.

На втором этапе форматирования создается очищенное дерево объектов форматирования (рис. ). Процесс очистки производит отображение из свойств в признаки (traits).

Это процесс состоит из:

* развёртывания сокращений (shorthand) в конкретные свойства;
* отображения соответствующих свойств;
* определения вычисленных значений (могут выполняться вычисления выражений, например, преобразования относительных единиц в абсолютные);
* учета обработки пробелов (свойство white-space-treatment) и перехода на новую строку (свойство linefeed-treatment);
* учета наследования свойств.

Очистка

Объект форматирования

**fo:block**

Свойства

**start-indent="2em"**

**font-size="20pt"**

Очистка

Объект форматирования

**fo:block**

Признаки

**start-indent="40pt"**

**font-size="20pt"**

а)

б)

Дерево объектов

форматирования и свойств

Дерево объектов

форматирования и признаков

Рис. Второй этап форматирования: а) всего документа;

б) отдельного элемента и свойства

На третьем этапе форматирования выполняется конструирование дерева геометрических областей, называемого деревом областей (area tree) (рис. ). Геометрические области образуют последовательность из одной или нескольких страниц. Каждая геометрическая область содержит позицию на странице, описание компоновки области и может иметь фон, заполнение и рамку. Например, форматирование одиночного символа генерирует область, в которой размещается так называемый глиф (glyph), используемый для визуального представления символа. Области могут вкладываться одна в другую.

При отображении берётся дерево областей, абстрактная модель представления (в терминах страницы и коллекции её областей) и производится вывод на определённый носитель: экран дисплея, лист бумаги или экран мобильного телефона.

Создание дерева областей

Объект форматирования

**fo:block**

Признаки

**start-indent="2em"**

**font-size="20pt"**

Область

**block-area**

Признаки

**start-indent="40pt"**

**font-size="20pt"**

а)

б)

Дерево объектов

форматирования и признаков

Визуализированные

признаки дерева областей

Создание дерева областей

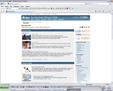


Рис.. Третий этап форматирования: а) всего документа;

б) отдельного элемента и признака

### Оформление страниц в XSL

#### Структура страницы

Структура объектов форматирования для вывода страниц представлена на рис.

Корневым элементом документа FO является элемент fo:root. Содержимым элемента является один элемент layout-master-set, 0 или 1 элементов declarations и один и более элементов page-sequence.

Элемент fo:root имеет единственное свойство media-usage. Это свойство используется для управления тем, как выбранный носитель вывода, специфицированный таблицей стилей, используется для представления страницы (страниц) и может принимать одно из следующих значений:

* "auto" – пользовательский агент (например, Web-браузер) сам определяет, какое значение свойства (отличное от "auto") использовать (значение по умолчанию);
* "paginate" – последовательность страниц генерируется из объектов fo:page-sequence, которые являются потомками объекта fo:root;
* "bounded-in-one-dimension" – генерируется только одна страница на объект fo:page-sequence, причем на первой используемой мастер-странице должно быть задано только "page-height" или только "page-width";
* "unbounded " – генерируется только одна страница на объект fo:page-sequence, причем на первой используемой мастер-странице не должно быть задано ни "page-height", ни "page-width".

Элемент fo:root содержит один дочерний элемент fo:layout-master-set, необязательный элемент fo:declarations и один и более элементов fo:page-sequence.

**fo:root**

**fo:layout-master-set**

**fo:declarations**

**fo:page-sequence**

**fo:simple-page-master**

**fo:page-sequence-master**

**fo:color-profile**

**fo:title**

**fo:static-content**

**fo:region-body**

**fo:region-before**

**fo:region-after**

**fo:region-start**

**fo:region-end**

**fo:single-page-master-reference**

**fo:repeatable-page-master-reference**

**fo:repeatable-page-master-alternatives**

**fo:conditional-page-master-reference**

**fo:flow**

Рис. Структура объектов форматирования для вывода страниц

#### Элементы fo:declarations и fo:color-profile

Элемент fo:declarations используется для группирования глобальных объявлений в таблице стилей. Этот элемент может содержать один или несколько элементов fo:color-profile. Для этого пустого элемента определены следующие свойства (в спецификации XSL атрибуты XML называются свойствами, поэтому в дальнейшем будет использоваться этот термин):

* src – URI для изображения или данных цветового профиля;
* color-profile-name – имя профиля цвета для внутренних ссылок;
* rendering-intent – профиль цвета, отличный от профиля по умолчанию (например, CMYK).

#### Элементы создания последовательности страниц

Элемент fo:page-sequence используется для определения создания последовательности или подпоследовательности страниц внутри документа: например, главы книги.

Содержимое этих страниц получается из потомков потока, состоящих из объектов потока: одного элемента fo:flow и любого количества элементов fo:static-content – дочерних элементов fo:page-sequence. Кроме того, на странице может быть элемент title. Структура этих страниц берется из элементов fo:page-sequence-master или их дочерних элементов, на которые ссылается свойство master-reference в fo:page-sequence. Последовательности областей, возвращаемых каждым объектом потока – потомком fo:page-sequence, становятся потомками генерируемых страниц, как описано далее.

Для элемента fo:page-sequence определены следующие свойства:

* master-reference – определяет имя головной страницы, используемой для создания страниц последовательности;
* country – страна (двухсимвольный идентификатор страны, либо значение none, если страна не имеет значения, либо значение inherit – наследование из родительского элемента) (по умолчанию none);
* language – язык (идентификатор языка, либо значение none, если язык не имеет значения, либо значение inherit – наследование из родительского элемента) (по умолчанию none);
* format – строка преобразования номера страницы в строку (задается аналогично параметру строка-формата функции format-number() в XSLT);
* letter-value, grouping-separator и grouping-size – действуют для номера строки аналогично одноименным атрибутам элемента xsl:number в XSLT;
* id – уникальный идентификатор данного элемента;
* initial-page-number и force-page-count – устанавливают соответственно номер первой страницы в последовательности страниц и ограничения на количество страниц в последовательности страниц.

Свойство initial-page-number может принимать следующие значения:

* "auto" – начальное значение будет установлено в 1, если в документе отсутствует предыдущий fo:page-sequence, иначе начальный номер будет больше, чем последний номер предшествующей последовательности;
* "auto-odd" – значение определяется тем же способом, что и для "auto", но, если значение четное, добавляется 1;
* "auto-even" – значение определяется тем же способом, что и для "auto", но, если значение нечетное, добавляется 1;
* "число" – положительное целое число, определяющее начальный номер последовательности страниц.

Свойство force-page-count может принимать следующие значения:

* "auto" – делает последнюю страницу в этой последовательности страниц нечетной, если начальный номер следующей последовательности страниц – четный, и наоборот;
* "odd" – делает номера последовательности страниц четными;
* "even" – делает номера последовательности страниц нечетными;
* "end-on-odd" – делает последнюю страницу последовательности страниц четной;
* "end-on-even" – делает последнюю страницу последовательности страниц нечетной;
* "no-force" – не делает номера последовательности страниц ни четными, ни нечетными.

Элемент fo:flow определяет последовательность областей, определяемых блоковыми элементами fo:block, fo:block-container, fo:table-and-caption, fo:table и fo:list-block – потомками данного элемента (может быть один или несколько таких элементов).

Единственным свойством элемента fo:flow является свойство flow-name, задающее уникальное имя в пределах последовательности fo:page-sequence.

Элемент fo:static-content содержит блоковый элемент или последовательность блоковых элементов, которые представлены в одном регионе или повторяется в одинаково именованных регионах на одной или более страницах в последовательности fo:page-sequence (обычно используется для повторяющихся или статичных верхних или нижних колонтитулов). Элемент также имеет свойство flow-name, задающее уникальное имя в пределах последовательности fo:page-sequence.

Элемент fo:title используется для связи заголовка с данной последовательностью страниц. Этот заголовок может использоваться интерактивным пользовательским агентом для идентификации страниц. Элемент может иметь либо строковое содержимое (#PCDATA), либо его содержимым могут быть внутристроковые элементы.

Для элемента fo:title можно задавать позаимствованные из CSS2 общие свойства фона (background-attachment, background-color, background-image, background-repeat, background-position-horizontal и background-position-vertical), рамки (border-before-color, border-before-style, border-before-width, border-after-color, border-after-style, border-after-width, border-start-color, border-start-style, border-start-width, border-end-color, border-end-style, border-end-width, border-top-color, border-top-style, border-top-width, border-bottom-color, border-bottom-style, border-bottom-width, border-left-color, border-left-style, border-left-width, border-right-color, border-right-style и border-right-width) и отступа (padding-before, padding-after, padding-start, padding-end, padding-top, padding-bottom, padding-left и padding-right).

Для элемента fo:title могут быть заданы также позаимствованные из CSS2 общие свойства шрифтов: font-family, font-size, font-stretch, font-size-adjust, font-style, font-variant и font-weight, а также следующие свойства блока (позаимствованные из CSS2): margin-top, margin-bottom, margin-left и margin-right.

Внутристроковые свойства space-start и space-end представляют собой составные свойства, задающие минимальные, оптимальные и максимальные значения для пространства перед любыми областями, генерируемыми элементом, а также обусловленность и приоритетность для этого пространства. Минимальные (space-start.minimum и space-end.minimum), оптимальные (space-start.optimum и space-end.optimum) и максимальные значения (space-start.maximum и space-end.maximum) задаются в фиксированных единицах измерения или в процентах. Обусловленность (space-start.conditionality и space-end.conditionality) – это перечислимое значение, которое управляет воздействием свойства space в начале или в конце внутристроковой области. Возможные значения: retain и discard; условный спецификатор пространства – это тот, для которого данное значение – discard.

Приоритет (space-start.precedence и space-end.precedence) имеет значение целое число – значение приоритета или специальную лексему force.

По умолчанию свойства space-start и space-end для минимального, оптимального и максимального значение равны "0pt", для обусловленности – "discard" и для приоритета – "0".

Свойство color определяет цвет заголовка и задается также, как в CSS2 – указанием имени цвета (например, "red") или вызовом функции rgb() (например, "rgb(255,0,0)").

Свойства line-height и visibility, позаимствованные из CSS2, определяют соответственно высоту строки и видимость элемента.

#### Элементы fo:layout-master-set и fo:simple-page-master

Элемент fo:layout-master-set является оболочкой вокруг всех головных наборов, используемых в документе: головных наборов последовательности страниц, головных наборов страниц и головных наборов регионов. Его содержимым является один или несколько элементов simple-page-master, либо элементов page-sequence-master.

Элемент fo:layout-master-set не генерирует области непосредственно. Генерация страниц выполняется с помощью элемента fo:page-sequence.

Элемент fo:simple-page-master используется при генерации страниц и определяет структуру страницы. Его содержимым является один элемент region-body, а также необязательные элементы region-before, region-after, region-start и region-end (по одному элементу для каждого региона). Страница может подразделяться на регионы (до пяти): region-body, region-before, region-after, region-start и region-end.

Так же, как и его родительский элемент, fo:simple-page-master не генерирует области непосредственно, а используется элементом fo:page-sequence для генерации страниц.

Для элемента fo:simple-page-master определены общие свойства полей блока, а также следующие свойства: master-name, page-height, page-width, reference-orientation, writing-mode.

Общие свойства полей блока включают позаимствованные из спецификации CSS свойства полей документа: margin-top, margin-bottom, margin-left и margin-right, свойства start-indent и end-indent, а также свойства space-before и space-after.

Свойства start-indent и end-indent определяют расстояние от начального края или конечного края прямоугольника содержимого содержащей области reference-area до начального края или конечного края прямоугольника содержимого данной блоковой области. Значение задается либо в фиксированных единицах измерения (в качестве единиц измерения используются те же единицы, что и в CSS: cm – сантиметры, mm – миллиметры, in – дюймы, pt – пункты, pc – пики, px – пиксели и em – высота текущего шрифта), либо в процентах.

Свойства space-before и space-after представляют собой составные свойства, задающие минимальные, оптимальные и максимальные значения для пространства перед областями before и after, генерируемыми данным элементом, и обусловленность и приоритетность для этого пространства. Они задаются аналогично свойствам space-start и space-end элемента fo:title.

Свойство master-name задает уникальное для потомков набора fo:layout-master-set имя. На это имя последовательно ссылаются как на значение свойств элементов fo:single-page-master-reference, fo:repeatable-page-master-reference и fo:conditional-page-master-reference при запросах этого элемента в процессе создания экземпляра страницы. Оно может использоваться также в объекте fo:page-sequence для задания набора, используемого при создании экземпляров страниц.

Свойства page-height и page-width определяют высоту и ширину страницы. Эти свойства могут иметь одно из следующих значений:

* "auto" – параметр определяется либо по размеру окна пользовательского агента, либо по размеру носителя (значение по умолчанию);
* "indefinite" – высота страницы определяется размером выводимого содержимого (высота и ширина страницы не могут иметь одновременно значение "indefinite");
* "размер" – определяет фиксированную высоту или ширину страницы в фиксированных единицах измерения;
* "inherit" – наследуется из вышележащего элемента.

Свойство reference-orientation определяет «вершину» для области содержимого по отношению к содержащей ее области. Это свойство имеет фиксированные значения "0" | "90" | "180 "| "270" | "-90" | "-180" | "-270" или "inherit". Числовые значения означают поворот области по отношению к содержащей ее области на заданное количество градусов по часовой стрелке (числа без знака) или против часовой стрелки (отрицательные числа). По умолчанию, значение этого свойства равно "0".

Свойство writing-mode определяет базовые режимы письма, используемые в различных языках. По умолчанию значение этого свойства "lr-tb": текст в строке записываются слева направо, а строки и блоки размещаются сверху вниз. Это свойство следует задавать только в тех случаях, когда используется другой порядок записи, например для японского или арабского языков.

#### Элементы fo:region-body, fo:region-before, fo:region-after, fo:region-start и fo:region-end

Когда для генерации страницы используется элемент fo:simple-page-master, генерируется пара порт-просмотра/ссылка, состоящая из области порта просмотра и области ссылки на страницу. Область порта просмотра (page-viewport-area) представляет физические границы носителя вывода. Область ссылки на страницу (page-reference-area) представляет часть страницы, на которой предполагается разместить содержимое, т.е. область в рамках полей страницы.

Такие же пары генерируются и для дочерних элементов fo:simple-page-master: fo:region-body, fo:region-before, fo:region-after, fo:region-start и fo:region-end (это пустые элементы без содержимого).

Расположение этих регионов на странице представлено на рис.

**Region-before**

**Region-after**

**Region-start**

**Region-end**

**Region-body**

Рис. Расположение регионов на странице

Для всех элементов регионов, так же как и для элемента fo:title, можно задавать позаимствованные из CSS2 общие свойства фона, рамки и отступа.

Кроме того, для всех элементов регионов определены свойства display-align, region-name, overflow, clip, reference-orientation и writing-mode.

Свойство display-align определяет вертикальное выравнивание региона. Допустимыми значениями этого свойства являются: "before" – выравнивание к верхнему краю, "after" – выравнивание к нижнему краю, "center" – выравнивание по центру, "auto" – для регионов совпадает со значением "before" (значение по умолчанию).

Свойство region-name используется для идентификации региона внутри элемента fo:simple-page-master. Каждый регион имеет имя по умолчанию – это имя региона с префиксом "xsl-", например, "xsl-region-body". Региону можно присвоить любое другое имя (но не имя другого региона по умолчанию). Обычно такое присваивание выполняется, когда нужно отличить четные и нечетные страницы.

Позаимствованное из CSS2 свойство overflow определяет, как будет вести себя регион, когда он не помещается в отведенную ему область. Значение "visible" определяет, что регион будет «наползать» на другие регионы, значение "hidden" определяет, что содержимое региона будет усекаться, причем размер и форма усекаемого региона определяются свойством clip, значение "scroll" также определяет, что содержимое региона будет усекаться, но должен использоваться механизм прокрутки и, наконец, значение "auto" (значение по умолчанию) определяется пользовательским агентом (обычно это значение совпадает со значением "scroll").

Значением свойства clip (также позаимствованного из CSS2) является вызов функции

"rect(сверху справа снизу слева)",

определяющей размер усечения (в пикселях) сверху, справа, снизу и слева.

Кроме описанных выше свойств, элементы fo:region-before, fo:region-after, fo:region-start и fo:region-end имеют общее свойство extent, определяющее (в фиксированных единицах длины или процентах) высоту или ширину региона (значение по умолчанию "0.0pt"), а элементы fo:region-before, fo:region-after имеют еще и свойство precedence, которое определяют порядок пересечения регионов в углах. Если это свойство имеет значение false (значение по умолчанию), то регионы start и end перекрывают соответствующий регион по углам, если true, то, наоборот, соответствующий регион перекрывает регионы start и end, как показано на предыдущем рисунке.

Элемент fo:region-body имеет следующие дополнительные свойства:

* общие свойства полей блока, совпадающие с общими свойствами полей элемента fo:simple-page-master;
* свойство column-count – положительное целое число, определяющее количество столбцов в регионе (по умолчанию равно "1");
* свойство column-gap – определяет расстояние (в фиксированных единицах измерения или процентах) между столбцами в регионе (по умолчанию равно "12.0pt").

Пример использования регионов при определении четных и нечетных страниц документа:

<fo:layout-master-set>

<fo:simple-page-master master-name="odd"

page-height="29.7cm" page-width="21cm"

margin-top="2.5cm" margin-bottom="2.5cm"

margin-left="3cm" margin-right="1.5cm">

<fo:region-body region-name="xsl-region-body"

margin-top="1cm" margin-bottom="1cm"

margin-left="1cm" margin-right="1cm"/>

<fo:region-before extent="1cm"

region-name="odd-before"/>

<fo:region-after extent="1cm"/>

<fo:region-start extent="1cm"/>

<fo:region-end extent="1cm"/>

</fo:simple-page-master>

<fo:simple-page-master master-name="even"

page-height="29.7cm" page-width="21cm"

margin-top="2.5cm" margin-bottom="2.5cm"

margin-left="1.5cm" margin-right="3cm">

<fo:region-body

margin-top="1cm" margin-bottom="1cm"

margin-left="1cm" margin-right="1cm"/>

<fo:region-before extent="1cm"

region-name="odd-before"/>

<fo:region-after extent="1cm"/>

<fo:region-start extent="1cm"/>

<fo:region-end extent="1cm"/>

</fo:simple-page-master>

</fo:layout-master-set>.

#### Элементы, формирующие последовательности страниц

Элемент fo:page-sequence-master и его дочерние элементы используются для генерации различных последовательностей страниц. Эти элементы не генерируют области непосредственно, а используются элементом fo:page-sequence для генерации страниц.

Содержимым элемента fo:page-sequence-master является один или несколько элементов single-page-master-reference, repeatable-page-master-reference и repeatable-page-master-alternatives.

Свойство master-name элемента fo:page-sequence-master должно совпадать с одним из уникальных имен, определенных в свойстве master-name наборов fo:layout-master-set.

Элемент fo:single-page-master-reference используется для формирования простейшей последовательности страниц, состоящей из одного экземпляра головной страницы.

Пустой элемент fo:repeatable-page-master-reference используется для формирования последовательности страниц, состоящей из повторяющихся экземпляров головной страницы.

Пустой элемент fo:repeatable-page-master-alternatives используется для формирования последовательности страниц, состоящей из повторяющихся экземпляров набора альтернативных головных страниц. Потомками этого элемента являются один или несколько элементов fo:conditional-page-master-reference.

Пустой элемент fo:conditional-page-master-reference используется для идентификации головной страницы при выполнении некоторого условия. Это позволяет использовать разные головные страницы, например, для четных и нечетных страниц, для первой страницы последовательности страниц или для пустых страниц.

Ссылка на значение master-name одного из наборов fo:layout-master-set задается в свойстве master-reference элементов fo:single-page-master-reference, fo:repeatable-page-master-reference и fo:conditional-page-master-reference при запросах этого элемента в процессе создания экземпляра страницы.

Для элементов fo:repeatable-page-master-reference и fo:repeatable-page-master-alternatives можно задать свойство maximum-repeats, задающее максимальное количество страниц в последовательности. Это свойство может быть целым числом, большим или равным 0, либо иметь значение no-limit для неограниченного количества страниц (значение по умолчанию).

Пример использования элемента fo:simple-page-master-reference:

<fo:root>

<fo:layout-master-set>

<fo:simple-page-master master-name="single"

page-height="12cm" page-width="10cm"

margin-top="1.5cm" margin-bottom="1.5cm"

margin-left="1cm" margin-right="1cm">

<fo:region-body

margin-top="1cm" margin-bottom="1cm"/>

</fo:simple-page-master>

<fo:page-sequence-master master-name="single-page">

<fo:single-page-master-reference

master-name="single"/>

</fo:page-sequence-master>

</fo:layout-master-set>

<fo:page-sequence master-name="single-page">

<!--Содержимое -->

</fo:page-sequence>

</fo:root>.

Пример использования элемента fo:repeatable-page-master-reference:

<fo:root>

<fo:layout-master-set>

<fo:simple-page-master master-name="many"

page-height="12cm" page-width="10cm"

margin-top="1.5cm" margin-bottom="1.5cm"

margin-left="1cm" margin-right="1cm">

<fo:region-body

margin-top="1cm" margin-bottom="1cm"/>

</fo:simple-page-master>

<fo:page-sequence-master master-name="many-pages">

<fo:repeatable-page-master-reference

master-name="many" maximum-repeats="10"/>

</fo:page-sequence-master>

</fo:layout-master-set>

<fo:page-sequence master-name="many-pages">

<!--Содержимое -->

</fo:page-sequence>

</fo:root>.

Правило отбора в элементе fo:conditional-page-master-reference определяется значением свойства page-position: "first" (первая), "last" (последняя), "rest" (остальные), "any" – любые (значение по умолчанию). Еще одно правило отбора может быть задано с помощью свойства odd-or-even, которое может иметь следующие значения: "odd" (нечетная страница), "even" (четная страница) или "any" (любая страница – значение по умолчанию). И, наконец последнее свойство blank-or-not-blank используется для генерации дополнительных пустых страниц. Это свойство может иметь следующие значения: "blank" (пустая страница должна быть сгенерирована, например, для поддержки чередования чётности страниц в начале или в конце последовательности страниц), "not-blank" (страница должна быть сгенерирована, если она не пустая) и "any" (страница должна быть сгенерирована в любом случае – значение по умолчанию).

Пример использования элементов fo:repeatable-page-master-alternatives и fo:conditional-page-master-reference:

<?xml version="1.0" encoding="Windows-1251"?>

<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

<fo:layout-master-set>

<!-- layout for the first page -->

<!-- Вывод для нечетных страниц -->

<fo:simple-page-master master-name="odd"

page-height="29.7cm" page-width="21.0cm"

margin-top="2cm" margin-bottom="2cm"

margin-left="3.5cm" margin-right="1.5cm">

<fo:region-body

margin-top="2cm" margin-bottom="2cm"/>

</fo:simple-page-master>

<!-- Вывод для четных страниц -->

<fo:simple-page-master master-name="even"

page-height="29.7cm" page-width="21.0cm"

margin-top="2cm" margin-bottom="2cm"

margin-left="1.5cm" margin-right="3.5cm">

<fo:region-body

margin-top="2cm" margin-bottom="2cm"/>

</fo:simple-page-master>

<fo:page-sequence-master master-name="chapter">

<fo:repeatable-page-master-alternatives>

<fo:conditional-page-master-reference

master-name="odd"

page-position="rest"

odd-or-even="odd"/>

<fo:conditional-page-master-reference

master-name="even"

page-position="rest"

odd-or-even="even"/>

</fo:repeatable-page-master-alternatives>

</fo:page-sequence-master>

</fo:layout-master-set>

<fo:page-sequence master-name="run1"

initial-page-number="1">

<fo:flow flow-name="xsl-region-body">

<!--Содержимое -->

</fo:flow>

</fo:page-sequence></fo:root>.