Секреты XML-преобразований

2-е издание
Охватывает XSLT 2.0

XSLT

Символ

O'REILLY®

Дуг Тидуэлл

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 978-5-93286-150-9, название «XSLT, 2-е издание» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.

XSLT

Second Edition

Doug Tidwell

O'REILLY®

XSLT

Второе издание

Дуг Тидуэлл

Санкт-Петербург — Москва
2010

Дуг Тидуэлл

XSLT, 2-е издание

Перевод Е. Матвеева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Редактор	*О. Меркулова*
Корректор	*О. Макарова*
Верстка	*Д. Орлова*

Тидуэлл Д.

XSLT, 2-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2010. – 960 с., ил.

ISBN: 978-5-93286-150-9

Цель книги «XSLT», 2-е издание – научить читателя эффективно использовать XSLT. Рассматриваются как XSLT 1.0, так и XSLT 2.0, а также версии 1.0 и 2.0 языка XPath. В спецификации XSLT 2.0 язык был дополнен рядом важных возможностей, многие из которых связаны с изменениями в XPath 2.0. Эти два языка проектировались в расчете на совместное использование: XPath определяет преобразуемые части документа XML, а XSLT указывает, как должно быть выполнено преобразование.

В начальных главах рассматриваются основы XSLT, в том числе простые таблицы стилей и настройка процессоров, особое внимание уделяется преобразованиям на базе шаблонов, обсуждаются основные концепции XPath 1.0 и 2.0. Вы узнаете, как в XSLT 2.0 интегрирована поддержка XML Schema, научитесь определять элементы и типы данных и использовать их в своих таблицах стилей. В завершающих главах рассматриваются примеры, в которых задействованы все возможности XSLT и XPath. Вы увидите, как одна и та же задача решается в XSLT 1.0 и 2.0, что поможет решить, какая версия лучше всего подходит для вашего проекта. Вторая часть книги содержит приложения, в которых приводятся сотни таблиц стилей, в том числе примеры для каждого элемента, функции и оператора, определяемых в XSLT и XPath.

Во всех главах представлены решения стандартных задач, в которых XSLT проявляет свои чрезвычайно мощные и полезные возможности. Применение XSLT 2.0 во многих случаях существенно упрощает жизнь программиста. Книга написана как учебник для тех, кто начинает изучать XSLT, а опытные специалисты смогут использовать ее как справочник по функциям и различным аспектам языка.

ISBN: 978-5-93286-150-9
ISBN: 978-0-596-52721-1 (англ)

© Издательство Символ-Плюс, 2010

Authorized translation of the English edition © 2008 O'Reilly Media, Inc. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

Все права на данное издание защищены Законодательством РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 380-5007, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.

Подписано в печать 25.09.2009. Формат 70×100$^{1}/_{16}$. Печать офсетная.
Объем 60 печ. л. Тираж 1200 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Посвящается моей семье – моей замечательной жене Шери Кастл и изумительной дочери Лили – за их любовь, поддержку и понимание. Все, что я делаю, становится возможным и обретает смысл только благодаря им.

...и моя особая благодарность нашей собаке Домино, которая частенько самоотверженно просовывает свою мохнатую голову между моими руками и клавиатурой, защищая меня от запястно-лучевого синдрома. Хороший пес!

Оглавление

Предисловие 10

1. Первые шаги 19

Архитектура XSLT 19
Основы XML 23
Установка процессоров XSLT 41
Итоги 45

2. Обязательный пример «Hello World» 46

Преобразование Hello World 46
Как обрабатываются таблицы стилей 48
Структура таблицы стилей 51
Галерея преобразований 57
Итоги 65

3. XPath: описание иголки в стоге сена 67

Модель данных в XPath 68
Пути доступа 78
Шаблоны значений атрибутов 90
Типы данных 91
Операторы XPath 96
[2.0] Комментарии в выражениях XPath 129
[2.0] Типы процессоров XSLT 2.0 131
Представление документа XML в XPath 132
Итоги 140

4. Создание выходных данных 141

Построение текста 141
Нумерация 147
Форматирование чисел 156
[2.0] Форматирование даты и времени 159
Элементы <xsl:copy> и <xsl:copy-of> 162
Обработка пропусков 169
Итоги 174

5. Программная логика и управляющие элементы 175
Элементы программной логики в XSLT 176
Вызов шаблонов по имени 181
Параметры 183
Переменные 199
Рекурсия 201
Имитация цикла for 206
Итоги 211

6. Связывание и перекрестные ссылки 212
Типы данных XML ID, IDREF и IDREFS 212
Функция key() и элемент <xsl:key> 225
Генерирование ссылок в неструктурированных документах 230
Итоги 236

7. Сортировка и группировка элементов 237
Сортировка данных при помощи элемента <xsl:sort> 237
[2.0] Элемент <xsl:perform-sort> 248
Группировка узлов 252
[2.0] Новый синтаксис группировки в XSLT 2.0 261
Итоги 278

8. Объединение документов 279
Функция document() 279
Функция document() и сортировка 288
Реализация таблиц отображения 290
Группировка по нескольким документам 293
[2.0] Упрощение таблиц стилей в XSLT 2.0 295
[2.0] Функции doc() и doc-available() 305
[2.0] Функция collection() 307
[2.0] Функции unparsed-text() и unparsed-text-available() 309
Итоги 312

9. Расширение XSTL 313
Механизм расширения XSLT 314
[2.0] Создание новых функций с помощью элемента <xsl:function> 316
Пример: построение нескольких выходных файлов 318
Создание пользовательских схем упорядочения 323
Генерирование замаскированной графики 330
Пример: построение круговых диаграмм в формате SVG 340
Написание расширений на других языках 365
Использование функций расширения из библиотеки EXSLT 369
Обращение к базе данных из элемента расширения 372
Создание фотоальбома с помощью элементов расширения 379
Итоги 399

A. Справочник по элементам XSLT 400

B. Справочник по XPath 574

C. Справочник по функциям XSLT, XPath и XQuery 591

D. Краткий обзор XML Schema 878

E. [2.0] Регулярные выражения 903

F. Форматные коды XSLT 921

G. Руководство по переходу на XSLT 2.0 927

Глоссарий 933

Алфавитный указатель 945

Предисловие

О книге

Цель данной книги – научить читателя эффективно использовать XSLT (Extensible Stylesheet Language for Transformations). В ней рассматриваются как XSLT 1.0, так и XSLT 2.0, а также версии 1.0 и 2.0 языка XPath (XML Path Language). Эти два языка проектировались в расчете на совместное использование: XPath определяет преобразуемые части документа XML, а XSLT указывает, как должно быть выполнено преобразование.

Несколько начальных глав книги описывают возможности XSLT на примере решения стандартных задач. В завершающих главах приводится набор примеров, в которых задействованы все возможности XSLT и XPath. Книга написана как учебник для изучения языка читателями, не обладающими опытом работы на нем. А когда вы почувствуете себя более уверенно, сможете использовать книгу как справочник по функциям и различным аспектам языка.

Об авторе

Полагаю, прежде чем браться за дело, мне стоит немного рассказать о себе.

Верю в открытые решения, не зависящие от платформы и основанные на стандартах

Если какая-либо сторона вашей деловой жизни привязывает вас к закрытым, запатентованным, зависимым от конкретной платформы решениям, я рекомендую подумать об изменениях. Книга показывает, как взять данные под полный контроль и переместить их из одного места в другое на ваших условиях, а не на условиях, навязанных разработчиком вашего программного обеспечения. XML смещает баланс сил от разработчиков в сторону пользователей. Если ваш инструментарий создает неудобства в работе, если вы не можете получить свои данные в том виде и в тот момент, когда требуется, вам больше не придется мириться с этим.

Предполагаю, что у вас мало свободного времени

Лучшая рецензия к первому изданию данной книги начиналась словами: «Я никогда не стану читать эту книгу». На самом деле рецензия была вполне положительной, т. к. далее рецензент пояснял: «Когда у меня возникает проблема, я снимаю книгу с полки, открываю алфавитный указатель и через пять минут нахожу решение. Затем снова ставлю книгу на полку».

Именно такую книгу я пытался написать. В книге приводятся сотни таблиц стилей, в том числе примеры для каждого элемента и функции XSLT, для каждого оператора, определяемого в XSLT и XPath. В первых главах объясняется, как работают таблицы стилей и что необходимо для эффективной работы с XSLT. Когда вы усвоите эти основы, остальной материал сможете использовать как справочник.

Меня не интересует, каким стандартным инструментарием вы пользуетесь

Моя задача как автора и учителя – показать, как использовать стандартный инструментарий для облегчения жизни. Я не собираюсь продавать вам парсер, процессор XSLT, тостер или что-нибудь еще; используйте любые инструменты на свое усмотрение. Я рекомендую присмотреться ко всем инструментам и выбрать то, что вам больше всего понравится. Во время написания книги я использовал четыре процессора для тестирования примеров:

- Почти все примеры книги были протестированы в замечательном процессоре XSLT Saxon, который написал Майкл Кэй (Michael Kay). Версия Saxon с открытым исходным кодом поддерживает все спецификации XSLT 2.0, XPath 2.0 и XQuery 1.0 *кроме* схемоспецифических функций. Процессор доктора Кэя, использовавшийся в спецификации XSLT 2.0, в настоящее время является наиболее полной реализацией XSLT 2.0.

  Saxon-B (базовый процессор без поддержки схем) доступен по адресу *http://saxon.sourceforge.net/*. Страница проекта в SourceForge находится по адресу *http://sourceforge.net/project/saxon*. Saxon существует в версиях для Java и .NET.

  Имеется и коммерческая версия Saxon с полноценной поддержкой схем. За дополнительной информацией о Saxon-SA (схемосовместимая версия) обращайтесь по адресу *http://www.saxonica.com*.

- Во всех примерах XSLT 2.0 использовалось также ядро XSL от Altova XML Spy. Хотя Altova XSLT и не распространяется с открытым исходным кодом, но предоставляет полноценную поддержку схем в бесплатном продукте. Лицензия ядра Altova в настоящее время разрешает его вторичное распространение с вашим кодом. Чтобы загрузить ядро и узнать условия лицензии, посетите страницу *http://www.altova.com/altovaxml.html*.

- Процессор XSLT Xalan из проекта Apache поддерживает почти все примеры XSLT 1.0, приводимые в книге. (Не поддерживаются только таблицы стилей XSLT 1.0 с расширениями, написанными для других процессоров.) Этот процессор обладает прямой совместимостью, поэтому он может работать с таблицами стилей XSLT 2.0.

  Java-версия этого процессора, Xalan-J, доступна по адресу *http://xml.apache.org/xalan-j/*. Существует версия и для C++ по адресу *http://xml.apache.org/xalan-c/*.

- XSLT 1.0 поддерживается на платформе Microsoft .NET, а также процессором MSXSL. В этой книге платформе Microsoft было уделено существенно большее внимание, чем в предыдущем издании. Помимо тестирования всех примеров XSLT 1.0 в инструментарии Microsoft, здесь приводятся расширения XSLT, написанные на C# и EcmaScript.

  Процессор XSLT MSXSL доступен по адресу *http://msdn.microsoft.com/XML/XMLDownloads/default.aspx*. Имеется также процессор XSLT, встроенный в инфраструктуру .NET; он входит в пространство имен `System.Xml.Xsl`.

XSLT – инструмент, а не религия

Старая пословица гласит, что человеку с молотком все вокруг кажется гвоздем. Я не собираюсь утверждать, что XSLT решит все проблемы, с которыми вы когда-либо столкнетесь. В главе 1 обсуждаются причины создания XML и XSLT, а также решения, принятые при проектировании XSLT; я постарался представить типичные задачи, для решения которых создавался язык XSLT. Во всех главах книги приводятся стандартные ситуации, в которых XSLT проявляет свои чрезвычайно мощные и полезные возможности.

Но, несмотря на все сказанное, если какой-то инструмент справляется с задачей лучше, чем XSLT, не сомневайтесь и используйте его. Например, в XSLT имеются функции сортировки и группировки. Если преобразуемые данные берутся из реляционной базы данных, вероятно, функции `ORDER BY` и `GROUP BY` будут намного эффективнее сортировки и группировки средствами XSLT. Язык XSLT – мощное дополнение к вашему инструментарию, но это не значит, что все остальные инструменты следует выбросить.

Не переносите все таблицы стилей только потому, что существует новая версия XSLT

При выходе новой версии языка, стандарта или программного пакета всегда приходится решать, стоит ли переходить на нее, и этот выбор зависит от вашего приложения. Если вы строите веб-приложение, в котором броузер должен обрабатывать таблицы стилей XSLT на стороне клиента, переход на XSLT 2.0 станет возможен лишь после того, как

XSLT 2.0 будет поддерживаться всеми основными броузерами. Это произойдет нескоро. С другой стороны, если XSLT используется для преобразования данных и отправки преобразованных данных клиенту, немедленный переход на XSLT 2.0 вполне реален. За крайне немногочисленными исключениями все, что работало в XSLT 1.0, будет работать и в XSLT 2.0. Переход на XSLT 2.0 рассматривается в приложении G.

XSLT 2.0 и XPath 2.0 обладают множеством новых аспектов, которые упрощают написание и сопровождение таблиц стилей, а также расширяют их возможности. Вам несомненно стоит выделить время на то, чтобы познакомиться с этими возможностями и определить, какие из них пригодятся в вашей работе.

Структура книги

В спецификации XSLT 2.0 язык был дополнен рядом важных новых возможностей, многие из которых связаны с изменениями в XPath 2.0. Самые большие трудности у меня как у автора возникли с выбором структуры материала. Конечно, можно было написать книгу как учебник по XSLT 2.0, предполагая, что все перейдут на XSLT 2.0 при первой возможности. На мой взгляд, этого не произойдет, поэтому я отказался от такого пути. Я попытался изложить материал с точки зрения стандартных задач – тех, которые вам с большой вероятностью придется решать при помощи XSLT. Если в XSLT 2.0 имеются новые возможности, относящиеся к этим задачам, я рассказываю о них после описания тех концепций, на которых строятся таблицы стилей. Обычно XSLT 2.0 заметно упрощает жизнь, поэтому в самом начале изложения я упоминаю о том, что пользователям XSLT 2.0 доступно более простое решение.

Как и первое издание, данная книга состоит из двух частей: глав с описанием различных концепций и задач и серии приложений, которые образуют справочник по всем элементам, функциям, операторам и другим деталям, необходимым для написания таблиц стилей. Освоив XSLT, вы сможете использовать приложения как справочник по всему, что относится к XSLT и XPath.

Книга состоит из следующих глав:

Глава 1 «Первые шаги»

Изложение основ XML и обсуждение установки процессоров XSLT, используемых в книге.

Глава 2 «Обязательный пример Hello World»

Знакомство с XML-версией документа «Hello World» и рассмотрение таблиц стилей, преобразующих его в другие форматы.

Глава 3 «XPath: описание иголки в стоге сена»

Основы XPath – языка, используемого для описания частей документа XML. В этой главе углубленно рассматриваются многие изменения, появившиеся в XPath 2.0.

Глава 4 «Создание выходных данных»

Основы создания вывода, в том числе извлечение текста, копирование информации и нумерация.

Глава 5 «Программная логика и управляющие элементы»

Элементы логики XSLT (`<xsl:if>` и `<xsl:choose>`) и особенности их работы. Обсуждается новый оператор XPath 2.0 `if`.

Глава 6 «Связывание и перекрестные ссылки»

Различные способы построения ссылок между элементами в документах XML. Также рассматривается использование XPath для описания отношений между логически связанными элементами.

Глава 7 «Сортировка и группировка элементов»

Элемент `<xsl:sort>` и различные способы сортировки элементов в документе XML, новые средства группировки с использованием различных элементов и функций XSLT. В XSLT 2.0 группировка значительно упростилась; новые средства группировки также представлены в этой главе.

Глава 8 «Объединение документов»

Рассматривается функция `document()`, предназначенная для объединения нескольких документов XML, и написание таблиц стилей, работающих с коллекциями документов. Также представлены соответствующие функции XSLT 2.0.

Глава 9 «Расширение XSLT»

Рассматривается написание элементов и функций расширения. Несмотря на всю мощь и гибкость XSLT и XPath, в некоторых ситуациях требуется сделать что-то такое, что не может быть сделано средствами самого языка.

Вторая половина книги содержит справочную информацию:

Приложение A

Алфавитный список всех элементов XSLT с рекомендациями и примерами использования.

Приложение B

Сводка синтаксиса XPath, включающая типы данных, оси, типы узлов и операторы.

Приложение C

Алфавитный список всех функций, определяемых XPath и XSLT.

Приложение D

Краткий обзор XML Schema. Одним из нововведений XSLT 2.0 стала возможность применения XML Schema для определения типов данных и проверки структуры кода XML.

Приложение E

Синтаксис и возможности языка регулярных выражений, используемого в XPath 2.0 и XSLT 2.0.

Приложение F

Удобный список всех кодов форматирования XSLT и XPath.

Приложение G

Список факторов и основные методы перехода на XSLT 2.0.

Глоссарий

Термины, используемые в XSLT, XPath и XML вообще.

Условные обозначения

В книге используются следующие условные обозначения:

Курсив

Курсивом выделены команды, адреса электронной почты, важные термины при первом упоминании и URL-адреса.

`Моноширинный шрифт`

Литералы, константы, листинги и разметка XML.

`Моноширинный полужирный шрифт`

Данные, вводимые пользователем.

`Моноширинный курсив`

Текст, который должен заменяться пользовательскими значениями.

Полезные советы, рекомендации или общие замечания.

Предупреждения и предостережения.

[1.0]

Информация, содержащаяся в тексте, относится *только* к XSLT 1.0 и XPath 1.0.

[2.0]

Информация, содержащаяся в тексте, относится к новым возможностям XSLT 2.0 и XPath 2.0.

[2.0 – Schema]

Информация, содержащаяся в тексте, относится к схемосовместимым процессорам XSLT 2.0.

Как с нами связаться

Мы сделали все возможное для того, чтобы проверить приведенную в книге информацию, однако не можем полностью исключать, что некоторые аспекты XSLT изменятся (более того, в тексте могут встретиться ошибки!). Пожалуйста, сообщайте нам обо всех найденных ошибках, а также делитесь пожеланиями для будущих изданий по адресу:

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

800-998-9938 (США или Канада)

707-829-0515 (международные или местные звонки)

707 829-0104 (факс)

С комментариями и техническими вопросами по поводу книги обращайтесь по электронной почте:

bookquestions@oreilly.com

На сайте издательства имеется веб-страница книги со списками обнаруженных опечаток, примерами и планами будущих изданий. Страница доступна по адресу:

http://www.oreilly.com/catalog/9780596527211

За дополнительной информацией о книгах и по другим вопросам обращайтесь на сайт O'Reilly:

http://www.oreilly.com

Safari® Enabled

Если на обложке вашей любимой технической книги есть пиктограмма Safari® Enabled, это означает, что книга доступна через O'Reilly Network Safari Bookshelf.

Система Safari лучше обычных электронных книг. Это целая виртуальная библиотека с возможностью поиска по тысячам лучших технических книг, копирования/вставки примеров кода, загрузки глав и быстрого поиска ответов, когда вам потребуется самая точная и актуальная информация. Safari можно бесплатно опробовать на сайте *http://safari.oreilly.com*.

Благодарности для второго издания

Хочу поблагодарить Джени Теннисон (Jeni Tennison), старшего рецензента этого издания. Ее умение разглядеть суть проблемы и указать самое простое и элегантное решение просто поражает. Я набил себе шишку, читая ее комментарии и хлопая себя по лбу со словами «Ну конечно! И как я сразу этого не понял!» Джени, спасибо тебе.

Огромную помощь мне оказала и отличная рецензия Присциллы Уолмсли (Priscilla Walmsley), особенно в области приложений с описаниями всех элементов и функций XSLT, XPath и XQuery. В результате примеры и терминология в этих разделах стали более правильными и полезными.

Огромная благодарность Майклу Кэю (Michael Kay) за предоставленную копию Saxon-SA, использованную для тестирования схемных примеров. Все сообщество XSLT перед ним в огромном долгу – и за написанный им процессор Saxon, и за то, что его стараниями спецификация XSLT 2.0 стала более понятной, полной и свободной от ошибок.

Книга была написана полностью в DocBook – чрезвычайно мощной номенклатуре XML для издательских целей. В ходе работы с DocBook две книги оказали мне неоценимую помощь. Первая – «DocBook: The Definitive Guide» (O'Reilly), написанная Нормом Уолшем (Norm Walsh) и Леонардом Мюллером (Leonard Mueller); книга доступна по адресу *http://www.oreilly.com/catalog/docbook/chapter/book/docbook.html*. При желании узнать что-нибудь о DocBook, начинайте с нее. Сообщество Open Source создало также чрезвычайно сложный набор таблиц стилей XSLT, преобразующих DocBook в разнообразные форматы. В работе с DocBook XSL мне очень помогла книга Боба Стейтона (Bob Stayton) «DocBook XSL: The Complete Guide» (Sagehill Enterprises; см. *http://sagehill.net/book-description.html*). Спасибо всем троим замечательным авторам.

Хочу поблагодарить также тех людей, с которыми работал на протяжении нескольких последних лет. Группа IBM developerWorks попрежнему оказывает на меня огромное влияние. Я всегда буду считать себя частью семьи developerWorks. За время, проведенное в организации IBM Developer Skills, мне посчастливилось работать с чрезвычайно талантливыми людьми. Участникам группы платят за то, чтобы они делились как можно большим объемом знаний с окружающими, а также за разработку бесплатных программ для профессоров и студентов по всему миру. Наконец, я хочу поблагодарить коллег по организации IBM Software Group Strategy, где я работаю сейчас. Я счастлив снова работать для Дирка Никола (Dirk Nicol), отца developerWorks.

Я борюсь с искушением назвать и другие имена, потому что боюсь кого-нибудь забыть. Надеюсь, все вы знаете, как много значите для меня и как много я узнал от вас.

В завершение я хочу поблагодарить Саймона Сен-Лорана (Simon St.Laurent) за его руководство проектом второго издания. Мы оба беспокоились о том, как бы включить в книгу XSLT 2.0 и XPath 2.0, чтобы она при этом не превратилась в 5000-страничный том. К сожалению, я нередко испытывал терпение Саймона – работа над некоторыми частями книги занимала гораздо больше времени, чем мы оба рассчитывали. Саймон, ты лучше всех.

Благодарности для первого издания

Прежде всего хочу поблагодарить рецензентов книги. Дэвид Марстон (David Marston) из Lotus был ведущим рецензентом; Дэвид, спасибо тебе за комментарии, мудрость и знания. Я получил массу полезной информации и содействия от Тони Колла (Tony Colle), Славко Малесвича (Slavko Malesvic), доктора Джо Молиториса (Dr. Joe Molitoris), Шейна О'Доннела (Shane O'Donnell), Энди Пайпера (Andy Piper), Шриниваса Рамарао (Sreenivas Ramarao), Майка Райли (Mike Riley) и Уилли Уилера (Willie Wheeler). Благодаря вашим комментариям и стараниям книга стала существенно лучше.

Хочу также поблагодарить своих коллег из developerWorks, чья поддержка помогла мне отважиться на этот проект. Взяться за вторую работу «на полную ставку» было непросто, но их советы и понимание в то время, как я пытался сбалансировать свои обязанности, оказали бесценную помощь. Еще важнее то, что я работаю среди самых интересных, творческих и замечательных людей, которых встречал в своей жизни. Парни, вы действительно крутые.

Когда я бывал дома (в Роли, Северная Каролина), мой душевный комфорт зависел от консультантов-диетологов из пиццерии Schiano: «Эй, тебе как обычно?» (Небольшая пауза) «Да, отлично, спасибо». Ничто так не успокаивает нервы, как пара ломтиков пиццы. Если вы окажетесь неподалеку от Роли, я настоятельно рекомендую посетить это заведение.

Наконец, я хочу поблагодарить сотрудников O'Reilly, особенно Лори Петрицки (Laurie Petricky) и Саймона Сен-Лорана (Simon St.Laurent). Лори, спасибо, что убедила меня взяться за этот проект и не бросала меня, когда мне было особенно трудно выкроить время для работы. Саймон, я получал удовольствие от чтения твоих книг уже несколько лет; для меня было честью работать с тобой. Твое руководство, техническая квалификация, терпение и советы были бесценны.

Огромное спасибо вам всем!

1

Первые шаги

В этой главе рассматриваются основы XML и логические принципы, на которых строится архитектура XSLT и XPath. Речь также пойдет о других веб-стандартах и их связи с XSLT и XPath. Завершается глава кратким обсуждением настройки процессора XSLT, чтобы вы могли работать с примерами, описанными в книге.

Архитектура XSLT

Формат XML за рекордное время прошел путь от проекта до прочно устоявшейся технологии, которая у всех на слуху. Благодаря своей гибкости как языка представления структурированных данных он занял место общепринятого языка обмена данными. Его первые сторонники использовали для разбора и обработки XML-документов различные программные интерфейсы, такие как DOM (Document Object Model) и SAX (Simple API for XML). Но после того как XML занял свое место среди доминирующих технологий, стало ясно: средний веб-программист не станет влезать в код Java, Visual Basic, Perl и Python, чтобы поработать с документами. Возникла необходимость в гибком, мощном, но относительно простом языке, способном обрабатывать XML.

Короче говоря, миру понадобился XSLT.

XSLT (Extensible Stylesheet Language for Transformations, расширяемый язык стилей для преобразований) официально рекомендован комитетом W3C (World Wide Web Consortium). Он предоставляет гибкие, мощные средства для преобразования XML-документов в другие форматы, например в документ HTML, другой документ XML, файл PDF (Portable Document Format), файл SVG (Scalable Vector Graphics), файл VRML (Virtual Reality Modeling Language), плоский текстовый файл, файл JPEG, словом, практически в любой формат. Вы пишете

таблицу стилей XSLT, определяющую правила преобразования документа XML, а процессор XSLT делает все остальное.

Комитет W3C определил два семейства стандартов таблиц стилей. Самый старый и простой механизм CSS (Cascading Style Sheets) используется для определения различных свойств элементов разметки. Хотя CSS может использоваться с XML, чаще всего он применяется для стилевого оформления документов HTML. Да, я могу определить свойства CSS так, чтобы некоторые элементы выводились синим цветом, шрифтом с кеглем 58 пунктов или с полужирным начертанием. Все это, конечно, замечательно, но есть некоторые вещи, на которые механизм CSS не способен:

- CSS не может изменить порядок следования элементов в документе. Если вы захотите отсортировать некоторые элементы или отфильтровать их по некоторому свойству, CSS вам в этом не поможет.
- CSS не выполняет вычисления. Если вы захотите вычислить и вывести значение (например, просуммировать числовые значения всех элементов `<price>` в документе), CSS вам в этом не поможет.
- CSS не может объединять документы. Если вы захотите объединить 53 документа с заказами на покупку и напечатать сводку всех заказанных товаров, CSS вам в этом не поможет.

Не считайте этот раздел критикой CSS; XSLT и CSS проектировались для разных целей. Одним из стандартных применений XSLT является построение документов HTML, использующих CSS. В разделе «Представление XPath документа XML» главы 3 приведен пример, который при помощи XSLT генерирует классы CSS, а затем использует эти классы для форматирования элементов HTML.

XSLT создавался как более мощный и гибкий язык для преобразования документов. В этой книге мы рассмотрим все возможности XSLT и представим каждую из них в контексте практических примеров. При проектировании XSLT ставились следующие цели:

- Каждая таблица стилей XSLT должна быть документом XML. Это означает, что вы можете написать таблицу стилей, которая преобразует другую таблицу стилей в еще одну таблицу стилей. Подобное рекурсивное мышление весьма типично для XSLT.
- Язык XSLT основан на поиске по шаблону. Большая часть таблиц стилей состоит из правил (в XSLT они называются *шаблонами* (*templates*)), описывающих преобразования документов. В сущности, каждое правило говорит следующее: «Если будет обнаружена часть документа, которая выглядит так-то, ее следует преобразовать таким-то образом». Вероятно, такой подход отличается от программирования, которым вы занимались прежде.
- Код XSLT не должен иметь побочных эффектов. Иначе говоря, XSLT проектировался с учетом оптимизации, чтобы к нему можно было

одновременно применять несколько правил из стилевых таблиц. Самое серьезное следствие этого ограничения – невозможность изменения переменных. После того как переменная будет связана со своим значением, изменить ее значение уже не удастся; если бы переменные могли изменяться, обработка одного правила могла бы иметь побочные эффекты, влияющие на другие правила. Эта особенность почти наверняка отличается от всего программирования, которым вы занимались прежде.

На XSLT значительно повлияла архитектура *функциональных языков программирования*, таких как Lisp, Scheme и Haskell. В этих языках также используются неизменяемые переменные. Вместо шаблонов XSLT в функциональных языках программы определяются как серии функций, каждая из которых для четко определенного ввода генерирует четко определенный результат (свободный от побочных эффектов, разумеется). Конечной целью является выполнение инструкций конкретного шаблона XSLT без влияния на выполнение любого другого шаблона XSLT.

- Циклы в XSLT заменяются итерацией и рекурсией. Если переменные не могут изменяться, как организовать аналог цикла `for` или `do-while`? В XSLT применяются две эквивалентные конструкции: итерации и рекурсия. *Итерация* означает, что вы пишете шаблон XSLT, который говорит: «Найти все объекты, которые выглядят так-то, и сделать с каждым из них то-то». Впрочем, такой подход несколько отличается от принятой в процедурных языках конструкции `do-while` со смыслом: «Делать то-то, пока остаются элементы для обработки»; но в данном случае он делает именно то, что нужно.

  К рекурсии тоже необходимо привыкнуть. Если вам потребуется реализовать нечто в духе команды `for` (например, `for i=1 to 10 do`), рекурсия вам поможет. В книге приводятся многочисленные примеры использования рекурсии; за дополнительной информацией обращайтесь к разделу «Рекурсия может почти все» главы 5.

Итак, исходя из перечисленных целей проектирования, какими достоинствами обладает XSLT? Рассмотрим несколько сценариев:

- Веб-сайт должен поставлять информацию на различные устройства. Вам необходимо обеспечить поддержку обычных броузеров для настольных компьютеров, систем быстрого обмена сообщениями, мобильных телефонов и других устройств с низким разрешением и ограниченной функциональностью. Было бы очень удобно создать информацию в виде структурированных документов, а затем преобразовывать эти документы во все необходимые форматы.
- Вы обмениваетесь данными с другими организациями, но все они используют разные СУБД. Было бы очень удобно определить данные в стандартном формате XML, а затем преобразовать докумен-

ты, написанные в этом формате, в файлы импорта (код SQL, данные, разделенные запятыми, и т. д.).

- Чтобы не отставать от жизни, каждые несколько месяцев ваш сайт проходит полную визуальную переработку. Такие технологии, как вставки на стороне сервера (server-side includes) и CSS, помогают, но они не всесильны. Было бы очень удобно хранить данные в гибком формате, преобразуемом к любому нужному виду; это значительно упростило бы процесс переработки.
- Вы храните документы в нескольких форматах. Все документы обрабатываются компьютером, но писать общие программы для их разбора и обработки затруднительно. Было бы очень удобно объединить все документы в один формат, а затем сгенерировать сводки и отчеты на основе полученной коллекции. А еще лучше, если бы в отчеты можно было включать вычисляемые значения, автоматически генерируемые графики и форматирование для высококачественной печати.

В этой книге мы покажем, как подобные задачи решаются в XSLT. Многие главы посвящены конкретным операциям: сортировке, группировке, генерированию ссылок между блоками данных. Однако начинать следует с азов.

[2.0] Архитектура XSLT 2.0

XSLT 2.0 – существенное усовершенствование языка. XSLT 2.0 использует спецификацию XPath 2.0, которая сама претерпела ряд значительных изменений. Между выходом XSLT 1.0/XPath 1.0 и XSLT 2.0/XPath 2.0 прошло чуть более семи лет (с 16 ноября 1999 года до 23 января 2007 года). Были поставлены следующие два главных требования, обусловившие внушительный объем работы по созданию XSLT 2.0 и XPath 2.0:

Поддержка XML Schema

XSLT и XPath теперь поддерживают XML Schema; это означает, что узлы и переменные могут обладать типом данных. Мы определяем значение типа `xs:dateTime`, а процессор XSLT следит за тем, чтобы это требование выполнялось. Все процессоры XSLT 2.0 поддерживают основные типы данных XML Schema. *Схемосовместимый процессор* (schema-aware processor) также поддерживает пользовательские типы данных. Если у вас имеется тип данных с именем `purchaseOrder`, вы можете использовать схемосовместимый процессор для работы со значениями этого типа.

Интеграция с XQuery

Разработка XQuery началась в 1998 году, а версия 1.0 была официально рекомендована комитетом W3C 23 января 2007 года. XQuery 1.0 и XPath 2.0 используют общие модель данных, функ-

ции и операторы. Должно быть, координация действий рабочих групп XQuery, XPath и XSLT была весьма непростой задачей.

Но, как бы то ни было, родовые мучения XSLT 2.0 и XPath 2.0 остались позади, и теперь мы имеем более мощный язык преобразования документов. Изменения будут упоминаться там, где они важны для решения тех задач, которые вы, скорее всего, будете решать на XSLT. Все технические подробности изложены в приложениях.

Основы XML

Почти все, чем мы будем заниматься в книге, так или иначе связано с документами XML. Таблицы стилей XSLT и сами являются документами XML, предназначенными для преобразования документов XML в нечто иное. Если у вас еще нет опыта работы с XML, мы рассмотрим основные положения в этом разделе. За дополнительной информацией об XML обращайтесь к книге Эрика Т. Рэя (Erik T. Ray) «Learning XML»[1] (O'Reilly, 2001), а также к книге Эллиота Гарольда (Elliotte Harold) и Скотта Минса (Scott Means) «XML in a Nutshell»[2] (O'Reilly, 2001).

Происхождение XML

XML происходит от языка SGML (Standard Generalized Markup Language). SGML, созданный доктором Чарльзом Гольдфарбом (Charles Goldfarb) в 1970-х годах, широко используется в мощных издательских системах. К сожалению, субъективная сложность SGML помешала его повсеместному принятию в отрасли (SGML также расшифровывают как «Sounds Great, Maybe Later» – «Выглядит здорово; может быть, потом...»). Интерес к SGML усилился, когда Тим Бернерс-Ли (Tim Berners-Lee) создал HTML на базе SGML. Прошло совсем немного времени, и вся компьютерная отрасль стала применять язык разметки для построения документов и приложений.

Недостаток HTML заключается в том, что его теги ориентированы на взаимодействие между человеком и компьютером. Во время изобретения World Wide Web в конце 1980-х годов это было вполне нормально. Но по мере того как Всемирная паутина стала проникать во все аспекты нашей жизни, на HTML пришлось проделывать массу странных вещей. Всем нам доводилось строить страницы HTML с неуклюжими таблицами, 1-пиксельными GIF-файлами и прочей ерундой только для того, чтобы страница нормально смотрелась в броузере. Язык XML проектировался с целью избавить нас от мучений и вернуть в мир структурированных документов.

---

[1] Эрик Рэй «Изучаем XML». – Пер. с англ. – СПб.: Символ-Плюс, 2001.

[2] Эллиот Гарольд, Скотт Минс «XML. Справочник». – Пер. с англ. – СПб.: Символ-Плюс, 2002.

При всех своих ограничениях HTML остается самым популярным языком разметки из всех существующих. Раз он так популярен, зачем нужен XML? Рассмотрим следующий чрезвычайно содержательный элемент HTML:

```
<td>12304</td>
```

Что представляет этот таинственный фрагмент данных?

- Почтовый индекс городка Скенектади, штат Нью-Йорк?
- Количество лампочек, ежемесячно заменяемых в Лас Вегасе?
- Количество «Фольксвагенов», проданных в Гонконге за последний год?
- Вес стального моста Харбор-Бридж в Сиднее (в тоннах)?

Правильный ответ: а кто его знает. Этот глупый пример наглядно показывает, что в данных HTML отсутствует структура. Даже если мы приведем всю таблицу, понять ее смысл можно будет только при использовании интеллекта (того, что находится между ушами). Если бы вы видели, что эта ячейка находится рядом с другой ячейкой, содержащей текст «Скенектади», а в заголовке над таблицей написано «Почтовые индексы штата Нью-Йорк», то, будучи существом разумным, вы бы могли правильно интерпретировать содержимое ячейки. С другой стороны, если бы вам понадобилось написать программу, которая получает таблицу HTML и пытается определить, содержат ли ее ячейки почтовые индексы, ваша работа была бы, по меньшей мере, непростой.

Большинство страниц HTML преследует единственную цель: определение внешнего вида документа. Ветераны в области разметки знают, что такой подход к созданию контента крайне нежелателен. *Отделение содержимого от представления* – один из проверенных временем постулатов в издательской отрасли; к сожалению, большинство страниц HTML даже отдаленно не стремится к этому идеалу. Любой документ XML должен содержать информацию, размеченную тегами, которые описывают смысл всех фрагментов информации и отношения между ними. В представлении документа (также называемом *визуализацией* (*rendering*)) задействованы правила и решения, отделенные от самого документа. Знакомясь с десятками примеров документов и приложений, вы увидите, что откладывание решений, относящихся к визуализации, на как можно более позднее время открывает значительные преимущества.

Рассмотрим документ с разметкой:

```
<?xml version="1.0"?>
<postalcodes>
  <title>Most-used postal codes in November 2000</title>
  <item>
    <city>Schenectady</city>
```

```
    <postalcode>12304</postalcode>
    <usage-count>2039</usage-count>
  </item>
  <item>
    <city>Kuala Lumpur</city>
    <postalcode>57000</postalcode>
    <usage-count>1983</usage-count>
  </item>
  <item>
    <city>London</city>
    <postalcode>SW1P 4RG</postalcode>
    <usage-count>1722</usage-count>
  </item>
  ...
</postalcodes>
```

Хотя мы все еще находимся в области выдуманных примеров, реализовать поиск почтовых индексов в любом документе, использующем этот набор тегов (в отличие от тегов HTML `<table>`, `<tr>`, `<td>` и т. д.), не так уж сложно. Программа должна искать содержимое любых элементов `<postalcode>` в документе. (Не будем забегать вперед, но написание таблицы стилей XSLT для решения этой задачи займет не более 30 минут, включая 25-минутный отдых.) Хорошо спроектированный документ XML сам идентифицирует каждый фрагмент данных и моделирует отношения между этими данными. А это означает, что мы можем быть уверены в правильности обработки документа XML.

И снова ключевая идея заключается в отделении содержимого от представления. Наш документ XML четко обозначает фрагменты данных и помещает их в формат, удобный для разбора. В книге мы представим ряд способов преобразования документа XML в разнообразные форматы. В частности, элемент `<postalcode>12304</postalcode>` легко преобразуется в `<td>12304</td>`.

Правила документов XML

В продолжение нашего знакомства с основами XML скажу, что при создании документов XML необходимо помнить о нескольких правилах. Все таблицы стилей, создаваемые в книге, сами являются документами XML, поэтому правила документов XML распространяются на все наши примеры. Правила относительно просты, хотя в подавляющем большинстве документов HTML они не соблюдаются.

Важное замечание: в спецификации XML 1.0 четко указано, что при обнаружении документа XML, нарушающего правила, парсер должен выдать исключение и прекратить работу. Парсер не должен пытаться вычислить правильную структуру документа. Спецификация избегает повторения ошибок HTML, из-за которых средний броузер пытается воспроизводить многие неполноценные документы.

Документ XML должен содержаться в одном элементе

Первый элемент документа XML должен содержать весь документ. Он называется *элементом документа*, или *корневым элементом*. Если документ содержит более одного корневого элемента, парсер XML выдает исключение. Следующий документ XML вполне допустим:

```
<?xml version="1.0"?>
<greeting>
  Hello, World!
</greeting>
```

Говоря точнее, этот документ *корректен* (*well-formed*). Документы XML описываются как *корректные* и *действительные* (*valid*); вскоре мы определим оба эти термина. Напротив, следующий документ XML недопустим:

```
<?xml version="1.0"?>
<greeting>
  Hello, World!
</greeting>
<greeting>
  Hey, Y'all!
</greeting>
```

Документ содержит два корневых элемента, и парсер XML отказывается обрабатывать его. Учтите также, что объявление XML (`<?xml version="1.0"?>`; подробности чуть позже) элементом не является.

Все элементы должны иметь правильную вложенность

Если один элемент создается внутри другого, то и завершаться он должен внутри этого элемента. Броузер HTML преспокойно отобразит следующий документ:

```
<b>I really, <i>really</b> like XML.</i>
```

А парсер XML при виде этого документа выдаст исключение. Если вы хотите добиться аналогичного эффекта, придется использовать код следующего вида:

```
<b>I really, <i>really</i></b><i> like XML.</i>
```

Все атрибуты должны находиться в ограничителях

Атрибуты заключаются в апострофы или кавычки. Следующие два тега XML эквивалентны:

```
<a href="http://www.oreilly.com">
<a href='http://www.oreilly.com'>
```

Если потребуется определить атрибут, содержащий апостроф или кавычку, используйте один символ внутри, а другой в качестве ограничителя. Если атрибут должен содержать как апостроф, так и кавычку,

используйте заранее определенные *сущности*: " для кавычек и ' для апострофов:

```
<book title="XSLT, Second Edition" publisher="O'Reilly/">
<book title="XSLT, Second Edition" publisher='O&apos;reilly'/>
```

И еще одно замечание: XML не разрешает создавать атрибуты без значений. Иначе говоря, элементы HTML вида <ol compact> в XML недействительны. Чтобы закодировать этот элемент в XML, необходимо присвоить ему значение, например <ol compact="compact">. (Это правило действует и в XHTML.)

В тегах XHTML учитывается регистр символов

В HTML теги <h1> и <H1> эквивалентны. В XML – нет. Если вы попытаетесь завершить элемент <h1> парным тегом </H1>, парсер выдаст исключение.

Все конечные теги являются обязательными

Еще одна область, которую нарушает большинство документов HTML. Ваш броузер не обратит внимания на отсутствие тега </p> или </br>, а для парсера XML эти теги обязательны.

Пустые теги могут содержать конечный маркер

Другими словами, следующие два фрагмента XML считаются идентичными:

```
<lily age="13"></lily>
```

```
<lily age="13"/>
```

Обратите внимание: между начальным и конечным тегами в первом примере нет ничего, даже пробела; именно это обстоятельство и делает его пустым тегом.

Объявления XML

Некоторые документы XML начинаются с *объявления XML* – строки следующего вида:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Если кодировка не указана, парсер XML считает, что используется UTF-8 или UTF-16. UTF (Unicode Transformation Format) – стандарт Юникода, использующий разное количество байтов для представления практически всех символов и идеограмм из всех мировых языков. Учтите, что каждый парсер поддерживает свой набор кодировок; чтобы узнать, какие варианты возможны в вашем случае, обратитесь к документации парсера.

DTD (Document Type Definition) и XML Schema

Все правила, обсуждавшиеся до настоящего момента, распространяются на все документы XML. Для определения других ограничений для ваших документов XML можно также использовать документы DTD и XML Schema. DTD и Schema – это метаязыки для определения характеристик XML. Допустим, вы хотите указать, что любой документ XML с описанием заказа на покупку должен начинаться с элемента `<po>`. В свою очередь, элемент `<po>` содержит элемент `<customer-id>`, один или более элементов `<item-ordered>`, а также элемент `<order-date>`. Вдобавок каждый элемент `<item-ordered>` должен содержать атрибут `part-number` и атрибут `quantity`.

Ниже приведен пример документа DTD, определяющего перечисленные ограничения:

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

<!ELEMENT po (customer-id , item-ordered+ , order-date)>

<!ELEMENT customer-id (#PCDATA)>

<!ELEMENT item-ordered EMPTY>

<!ATTLIST item-ordered  part-number CDATA  #REQUIRED
                        quantity    CDATA  #REQUIRED >
<!ELEMENT order-date EMPTY>

<!ATTLIST order-date  day   CDATA  #REQUIRED
                      month CDATA  #REQUIRED
                      year  CDATA  #REQUIRED >
```

А вот как выглядит документ XML Schema, определяющий тот же тип документа:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="po">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="customer-id"/>
        <xsd:element ref="item-ordered" maxOccurs="unbounded"/>
        <xsd:element ref="order-date"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="customer-id" type="xsd:string"/>

  <xsd:element name="item-ordered">
    <xsd:complexType>
      <xsd:attribute name="part-number" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
```

```
            <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="quantity" use="required" type="xsd:integer"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="order-date">
    <xsd:complexType>
      <xsd:attribute name="day" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:positiveInteger">
            <xsd:maxInclusive value="31"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="month" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:positiveInteger">
            <xsd:maxInclusive value="12"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="year" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:gYear">
            <xsd:maxInclusive value="2100"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:complexType>
  </xsd:element>
  </xsd:schema>
```

Документы XML Schema обладают двумя важными преимуществами перед DTD:

Они способны определять типы данных и другие сложные структуры, которые затруднительно или невозможно реализовать в DTD.

В предыдущем примере для данных документов XML определялись различные ограничения. Мы задали, что атрибут `day` должен быть целым числом от 1 до 31, а атрибут `month` – целым числом от 1 до 12. Мы также использовали регулярное выражение, определяющее атрибут `part-number` как число из пяти цифр, за которым могут следовать необязательные дефис и код из четырех цифр. Ничто из этого нельзя сделать в DTD. Документы XML Schema заметно мощнее DTD; о том, на что они способны, рассказано в приложении D.

Документы XML Schema сами являются документами XML.

А раз они являются документами XML, то мы можем написать таблицу стилей XSLT для операций с ними. Например, с XML Schema удобно работать в графическом представлении; можно создать иерархическую диаграмму, которая показывает, какие элементы находятся внутри других элементов. XML Schema также поддерживает элементы `<xsd:annotation>` и `<xsd:documentation>`, позволяющие включать в схему любой необходимый объем документации. После этого по таблице стилей схема преобразуется в документ HTML или файл PDF, а отношения между элементами, атрибутами, типами данных и прочей информацией используются для получения структурированной информации.

Атрибут `<order-date>` лучше всего определяется при помощи типа данных XML Schema `xsd:date`:

```
<xsd:element name="order-date" type="xsd:date"/>
```

В DTD дату пришлось разделить на три части, чтобы ее можно было сортировать или форматировать разными способами. С типом данных `xsd:date` схема проверяет действительность даты, к тому же мы можем использовать различные функции для ее сортировки или форматирования. (Эти функции рассматриваются в разделе «[2.0] Форматирование даты и времени» в главе 4.)

Документы корректные и действительные

Любой документ, в котором выполняются все описанные правила, называется *корректным* (*well-formed*). Если в дополнение к этому документ XML содержит ссылки на набор правил, описывающих структуру документа (DTD или XML Schema), и эти правила соблюдаются, документ называется *действительным* (*valid*).

Все действительные документы являются корректными; с другой стороны, не все корректные документы действительны.

Учтите, что проверка XML Schema может выполняться частично; XML Schema позволяет определять части документа, которые вообще не должны проверяться. С другой стороны, проверка DTD считается неудачной, если какая-либо часть документа XML не соответствует DTD.

Теги и элементы

Многие программисты считают эти термины взаимозаменяемыми, но тег и элемент не одно и то же. *Тег* представляет собой текст в угловых скобках (< и >), включая сами скобки. Существуют начальные теги, конечные теги и пустые теги. Тег состоит из имени элемента и, если он является начальным или пустым тегом, необязательных атрибутов. (В отличие от некоторых языков разметки, конечные теги в XML не могут содержать атрибуты.) *Элемент* состоит из своего начального и конечного тегов и всего, что находится между ними: текста, других

элементов, комментариев, ссылок на сущности, инструкций по обработке и т. д.

Пространства имен

Остается рассказать о *пространствах имен* XML. Пространства имен создавались для того, чтобы парсер мог различать два тега с одинаковыми именами. Допустим, вы проектируете номенклатуру XML для Интернет-магазина по продаже книг. При отправке заказа покупателю почта требует, чтобы адрес покупателя был задан в определенном формате. Скорее всего, в обеих номенклатурах будет определяться элемент `<title>`. Наш элемент `<title>` относится к названию книги, а в формате данных почты элемент `<title>` обозначает приставку к имени покупателя (Mr., Ms., Mrs. и т. д.). В документе с заказом в формате XML упоминаются как книги, так и покупатели; чтобы различить два элемента `<title>`, мы воспользуемся пространствами имен. Пространства имен объявляются следующим образом:

```
<xyz xmlns:books="http://www.myco.com/books"
     xmlns:addr="http://www.usps.com/addresses">
```

В этом примере атрибут `xmlns:books` связывает префикс `books` с одним пространством имен, а атрибут `xmlns:addr` связывает префикс `addr` с другим пространством имен. Таким образом, элемент `title` из пространства имен `books` кодируется в виде `<books: title>`, а ссылка на элемент `title` из пространства имен `addr` имеет вид `<addr:title>`.

Я упоминаю здесь пространства имен главным образом из-за того, что все элементы XSLT, упоминаемые в книге, снабжаются префиксом пространства имен `xsl`. Все таблицы стилей, которые мы создадим, будут начинаться примерно так:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

(Разумеется, таблица стилей, использующая специфические возможности XSLT 2.0, начинается с `version="2.0"`.) Заголовок связывает префикс пространства имен `xsl` со строкой `http://www.w3.org/1999/XSL/Transform`. Конкретное значение префикса роли не играет; заголовок с таким же успехом может выглядеть так:

```
<?xml version="1.0"?>
<pdq:stylesheet version="1.0"
  xmlns:pdq="http://www.w3.org/1999/XSL/Transform">
```

Важна лишь строка, с которой связывается префикс пространства имен. Следует учитывать также, что все таблицы стилей XSLT используют префиксы пространств имен для обработки содержащихся в них элементов XML. По умолчанию все элементы, не использующие префикс пространства имен `xsl`, не обрабатываются, а напрямую записы-

ваются в дерево результатов. Мы более подробно обсудим эту тему по мере изложения материала.

[2.0] Типы данных

XSLT 2.0 предоставляет поддержку большинства типов данных, определяемых в XML Schema. XSLT 2.0 также определяет новые типы данных для отрезков времени. Например, можно определить переменную XSLT и указать для нее тип данных `xs:integer` или `xs:dateTime`. При использовании схемосовместимого процессора XSLT 2.0 можно определять собственные типы данных и использовать их точно так же, как любые типы данных, определяемые XML Schema и XSLT 2.0. Типы данных и схемы рассматриваются в главе 3.

Программные интерфейсы для работы с XML: DOM, SAX и другие

Два самых популярных API для разбора документов XML – DOM (Document Object Model) и SAX (Simple API for XML). DOM официально рекомендован комитетом W3C (см. *http://www.w3.org/TR/REC-DOM-Level-1*), а SAX – фактический стандарт, созданный Дэвидом Меггинсоном (David Megginson) и другими участниками из рассылки XML-DEV (*http://lists.xml.org/archives*). Оба API будут кратко описаны чуть позже. В книге они почти не используются, но знакомство с ними даст вам некоторое представление о том, как работают процессоры XSLT.

За информацией о стандарте SAX обращайтесь на сайт *http://www.saxproject.org/*. Если вам захочется больше узнать о рассылке XML-DEV, отправьте сообщение по адресу *mailto:xml-dev-subscribe@lists.xml.org*. Архивы рассылки XML-DEV хранятся на сайте *http://lists.xml.org/archives/xml-dev/*.

DOM

Стандарт DOM проектировался для построения представления документа в виде иерархического дерева. Напомню, что каждый документ XML должен содержаться в одном элементе, который становится корнем дерева. Спецификация DOM определяет несколько интерфейсов, нейтральных по отношению к языку реализации:

`Node`

Основной тип данных DOM. `Document`, `Element`, `Attr`, `Text`, `Comment` и `ProcessingInstruction` расширяют интерфейс `Node`.

`Document`

DOM-представление документа XML. По объекту `Document` можно найти корень дерева (элемент `Document`); перемещение по всему дереву от корня позволит найти все элементы, атрибуты, текст, комментарии, инструкции по обработке и т. д. в документе XML.

`Element`

Интерфейс представляет элемент в документе XML.

`Attr`

Интерфейс представляет атрибут элемента в документе XML.

`Text`

Интерфейс представляет блок текста из документа XML. Любой текст в документе XML представляется узлом `Text`. А это означает, что текст объекта DOM является потомком объекта, а не его свойством. Текст объекта `Element` представляется дочерним объектом `Text` объекта `Element`; текст объекта `Attr` также представляется подобным способом.

`Comment`

Интерфейс представляет комментарий в документе XML. Комментарии начинаются с последовательности `<!--` и завершаются последовательностью `-->`. Для их содержимого устанавливается единственное ограничение: два последовательных дефиса (`--`) могут находиться только в начале или в конце комментария. В остальном комментарий может содержать произвольные символы, включая угловые скобки (`<>`), амперсанды (`&`), апострофы и кавычки (`' "`).

`ProcessingInstruction`

Интерфейс представляет инструкцию по обработке в документе XML. Инструкции по обработке выглядят примерно так:

```
<?xml-stylesheet href="case-study.xsl" type="text/xsl"?>
```

Инструкции по обработке (также называемые PI, Processing Instruction) содержат информацию, относящуюся к конкретному процессору. Приведенная инструкция демонстрирует стандартный способ связывания таблицы стилей XSLT с документом XML (подробности чуть позже).

При разборе документа XML парсер DOM:

- Создает объекты (`Element`, `Attr`, `Text`, `Comment`), представляющие содержимое документа. Эти объекты реализуют интерфейсы, определенные в спецификации DOM.
- Строит из этих элементов дерево. Каждый объект `Element` в документе XML обладает свойствами (например, имя), а также может содержать дочерние объекты.
- Возвращает управление вашему коду только после завершения разбора всего документа. Это означает, что в больших документах разбор сопровождается значительной задержкой.

Самая важная особенность DOM заключается в том, что эта модель основана на древовидном представлении вашего документа. Процессор XSLT использует очень похожее древовидное представление (с небольшими различиями, например не все данные, с которыми мы имеем де-

ло в XPath и XSLT, имеют единый корневой элемент). Понимание принципов работы парсера DOM поможет вам лучше понять, как процессор XSLT обрабатывает ваш документ.

Пример дерева DOM. В DOM, XSLT и XPath используются древовидные структуры для представления данных из документа XML. По этой причине важно иметь хотя бы поверхностное представление о том, как DOM строит древовидную структуру. На рис. 1.1 изображено представление приведенного ранее документа `<postalcodes>` в виде дерева DOM.

Для выполнения таких операций, как поиск разных частей документа XML, сортировка поддеревьев по первому символу текста элемента `<postalcode>` или выделение поддеревьев, у которых текст элемента `<usage-count>` содержит числовое значение больше 500, необходимо начинать от корня дерева DOM и продвигаться вниз, перебирая потомков корневого элемента. При написании таблиц стилей XSLT мы тоже начинаем с корня дерева и продвигаемся вниз.

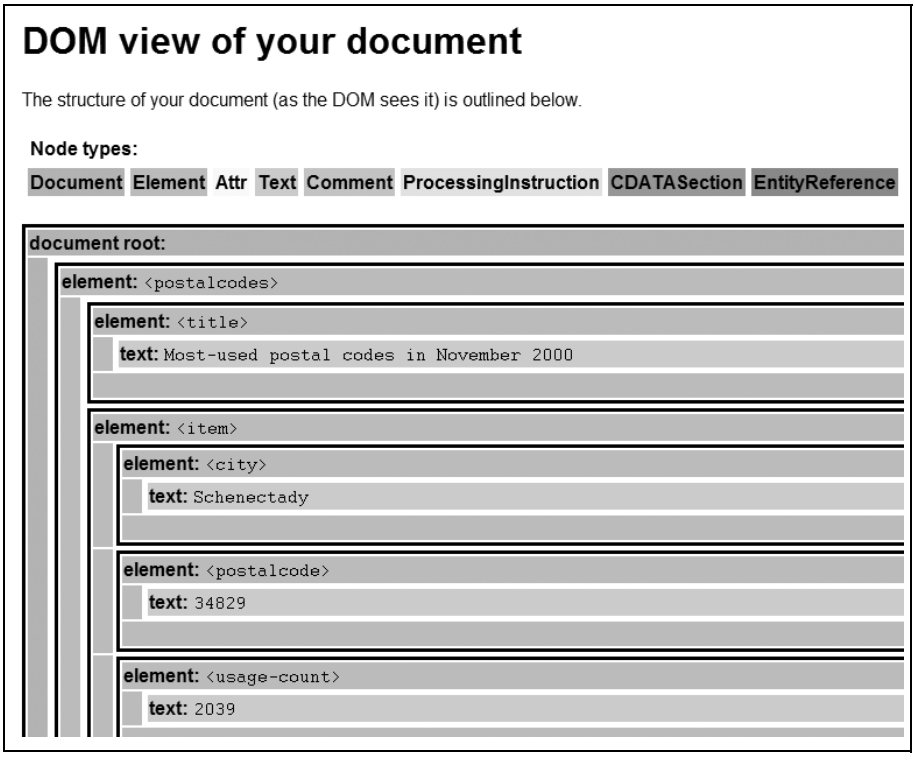


Рис. 1.1. *Представление документа XML в виде дерева DOM*

Откровенно говоря, дерево DOM для нашего документа сложнее, чем показано на красивой картинке. Пропуски в документе (переводы строк, табуляции, пробелы и т. д.) преобразуются в узлы `Text`. Обычно бывает желательно удалить эти пропуски, чтобы дерево DOM не загромождалось этими бесполезными узлами, но я оставил их, чтобы вы лучше почувствовали структуру документа XML.

SAX

Интерфейс SAX (Simple API for XML) был разработан Дэвидом Мегинсоном и другими участниками рассылки XML-DEV. Он обладает рядом важных отличий от DOM:

- SAX является интерактивным API. В процессе обработки документа парсер SAX отправляет события вашему коду. В отличие от DOM, вам не обязательно ждать, пока парсер обработает весь документ; события от парсера начинают поступать немедленно. Эти события позволяют узнать, когда парсер нашел начало документа, начало элемента, некий текст, конец элемента, инструкцию по обработке, конец документа и т. д.
- Проектировщики SAX постарались избежать больших затрат памяти, характерных для DOM. В мире SAX при обнаружении парсером различных частей документа XML вы получаете оповещения; сохранение полученной информации остается на ваше усмотрение. Если ничего не сделать с данными, найденными парсером, они просто пропадут.
- SAX, в отличие от DOM, не обеспечивает иерархическое представление документа. Если вам нужно получить разнообразную информацию о структуре документа XML и контексте заданного элемента, SAX особой пользы не принесет. События SAX не имеют состояния; иначе говоря, событие SAX не скажет: «А вот текст элемента `<postalcode>`, который упоминался ранее». Парсер SAX ограничится простым сообщением: «Вот какой-то текст». Если вам нужна информация о структуре документа XML, ее придется отслеживать самостоятельно.

Самое большое достоинство SAX – интерактивность. Большинство преобразований, выполняемых с использованием XSLT, происходит на сервере. На момент написания книги большинство процессоров XSLT базируется на парсерах DOM. Но в ближайшем будущем появятся процессоры XSLT, базирующиеся на парсерах SAX. Это означает, что процессор может приступать к построению результатов сразу же после начала разбора исходного документа, что способствует улучшению фактического быстродействия и ускорению обслуживания. Поскольку DOM, XPath и XSLT используют деревья для представления документов XML, модель DOM более важна для нашего обсуждения. И все же полезно знать, как работают парсеры SAX – особенно сейчас, когда процессоры XSLT на базе SAX постепенно начинают поднимать голову.

Другие программные интерфейсы

Помимо перечисленных существуют и другие программные интерфейсы XML, в том числе JDOM, DOM4J и StAX. Они обладают двумя важными характеристиками:

Строящие модель в памяти или управляемые событиями

Интерфейсы, строящие модель в памяти (такие как DOM), создают структуры данных, представляющие документ XML. Интерфейсы, управляемые событиями (такие как SAX), получают данные от парсера в процессе разбора документа.

Активные или пассивные

Активный (push) интерфейс «проталкивает» данные от парсера приложению. Когда у парсера появляются данные, он использует интерфейс обратного вызова для активной передачи этих данных приложению. SAX является примером активного интерфейса. С другой стороны, пассивный (pull) интерфейс тоже управляется событиями, но в этой модели приложение сообщает парсеру, что оно хочет получить следующее событие. StAX (Streaming API for XML) является примером пассивного интерфейса. (Интерфейс StAX также известен под названием JSR 173.)

Существуют еще два подхода, заслуживающие хотя бы краткого упоминания. В модели *привязки данных* документ XML преобразуется в объект. Содержимое исходного документа XML представляется свойствами этого объекта. Наконец, новая методология разбора, называемая *обработкой XML без извлечения*, создает виртуальные дескрипторы лексем, содержащие смещение, длину и прочую информацию о лексемах XML в самом файле XML.

В статье Википедии *http://en.wikipedia.org/wiki/XML#Processing_XML_files* приводится более подробная информация обо всех перечисленных методологиях, а также ссылки на различные инструменты, реализующие их.

Стандарты XSLT

XSLT 1.0 определяется в двух документах: спецификациях XSLT и XPath. С другой стороны, XSLT 2.0 и XPath 2.0 определяются в наборе из восьми документов. Каждая из этих спецификаций кратко обсуждается в следующем разделе.

XSL transformations (XSLT) version 1.0[1]

Исходный стандарт стал рекомендацией W3C 16 ноября 1999 г. Спецификация находится по адресу *http://www.w3.org/TR/xslt*.

[1] Названия спецификаций даются без перевода. – *Примеч. ред.*

XML path language (XPath) version 1.0

Спецификация XPath 1.0 стала стандартом в один день с XSLT 1.0. Изначально спецификация XPath была частью XSLT. Если вы собирались написать таблицу стилей для преобразования документа XML, вам был необходим синтаксис для описания различных частей этого документа. В ходе развития XSLT стало очевидно, что спецификация XPath может использоваться и для других целей, поэтому она была преобразована в отдельный стандарт. Определение XPath 1.0 можно найти по адресу *http://www.w3.org/TR/xpath*.

XSL transformations (XSLT) version 2.0

Базовое определение XSLT 2.0 находится по адресу *http://www.w3.org/TR/xslt20/*. В документе определяются элементы XSLT 2.0 и различные функции, а также механика обработки документов XML в XSLT 2.0.

XML path language (XPath) version 2.0

Базовое определение XPath 2.0 находится по адресу *http://www.w3.org/TR/xpath20/*. Спецификация XPath 2.0 построена на основе ряда других документов, перечисленных ниже.

XQuery 1.0 and XPath 2.0 Data Model (XDM)

Эта спецификация определяет способы организации данных в XPath 2.0, XSLT 2.0 и XQuery 1.0. В ней определяется информация, содержащаяся во входных данных процессора XSLT 2.0 или XQuery 1.0, а также все допустимые значения выражений XPath 2.0, XSLT 2.0 и XQuery 1.0. Спецификация находится по адресу *http://www.w3.org/TR/xpath-data-model/*.

XQuery 1.0 and XPath 2.0 functions and operators

Эта спецификация, также известная под сокращенным названием F&O, определяет все функции и операторы данных, доступные в XPath 2.0 и XQuery 1.0. Например, спецификация определяет, как `xs:yearMonthDuration` может делиться на значение `xs:double`. В ней определяется также функция `matches()`, которая устанавливает, совпадает ли значение с регулярным выражением. Спецификация находится по адресу *http://www.w3.org/TR/xpath-functions/*.

XQuery 1.0 and XPath 2.0 formal semantics

Спецификация формальной семантики определяет точный смысл всех допустимых выражений в XPath 2.0 и XQuery 1.0. В этих точных определениях используется спецификация «XQuery 1.0 and XPath 2.0 Data Model». Вероятно, это самая бесполезная спецификация для программистов XSLT; доступна по адресу *http://www.w3.org/TR/xquery-semantics/*.

XSLT 2.0 and XQuery 1.0 serialization

Спецификация определяет, каким образом выполняется сериализация экземпляров XQuery 1.0/XPath 2.0 Data Model. В примерах, приводимых в книге, мы обычно берем результаты, сгенерированные таблицей стилей XSLT, и записываем их в файл; данная спецификация определяет, как работает этот процесс. Спецификация находится по адресу *http://www.w3.org/TR/xslt-xquery-serialization/*.

XQuery 1.0: an XML query language

XQuery 1.0 – отдельный язык запросов, основанный на XPath и других языках запросов; является подмножеством XPath 2.0. В книге XQuery подробно не рассматривается, но следует помнить, что в XQuery используются модель данных, функции и операторы XPath 2.0. За полной информацией обращайтесь по адресу *http://www. w3.org/TR/xquery/*.

XML syntax for XQuery 1.0 (XQueryX)

Одним из требований к рабочей группе XQuery было использование в языке синтаксиса XML. Спецификация XQueryX предоставляет этот синтаксис. Грамматика XQuery отображается на теги XML; соответственно результат не очень понятен или удобен для человека, но очень полезен для различных служебных программ и утилит. Спецификация находится по адресу *http://www.w3.org/TR/xqueryx*.

Стандарты XML

При написании таблиц стилей используются два основных стандарта: XSLT и XPath. XSLT определяет набор примитивов для описания преобразования документа, а XPath определяет синтаксис описания различных мест в документах XML. При написании таблиц стилей XSLT передает информацию процессору о том, что следует сделать, а XPath – информацию о том, с каким документом это должно быть сделано. Оба стандарта доступны на сайте W3C; см. *http://www.w3.org/TR/xslt* и *http://www. w3.org/TR/xpath*.

Конечно, существуют и другие стандарты, относящиеся к XML. Они кратко перечислены в этом разделе, с короткими упоминаниями о том, в какой степени они относятся к нашей работе с XSLT и XPath.

XML 1.0

Фундамент, на котором строится все остальное. См. *http://www.w3.org/TR/REC-xml*.

XML 1.1

Стандарт XML 1.1 находится по адресу *http://www.w3.org/TR/xml11/*.

The Extensible Stylesheet Language (XSL)

Этот стандарт, также называемый *спецификацией форматирования объектов*, или *XSL-FO*, относится к визуализации элементов XML. Хотя большинство людей рассматривает визуализацию как форматирование для вывода в броузере или печати на принтере, ученые используют эту спецификацию для вывода элементов XML в письменности Брайля или в виде аудиофайлов. (Впрочем, главной областью применения этой технологии все же является получение высококачественного печатного вывода.) На момент написания книги последней версией XSL была версия 1.1. В паре примеров книги используются объекты форматирования и утилита Formatting Object to PDF translator (FOP) из проекта Apache XML Project; за дополнительной информацией о FOP обращайтесь на сайт *http://xml.apache.org/*. Дополнительную информацию об XSL можно найти по адресу *http://www.w3.org/TR/xsl*.

XML Schema

Использование XML Schema уже было продемонстрировано в одном из коротких примеров. Часть 1 этой спецификации описывает структуру документов XML; она содержит элементы XML, определяющие возможное содержимое документа XML. При помощи этих элементов можно указать, какие элементы должны быть вложены в другие элементы, сколько раз в документе может встречаться тот или иной элемент, какие атрибуты он может иметь и т. д. Часть 2 спецификации определяет основные типы данных, используемые в документах XML Schema, и правила создания новых типов данных на основе существующих.

Спецификации доступны по адресам *http://www.w3.org/TR/xmlschema-1* и *http://www.w3.org/TR/xmlschema-2*. Если вам понадобится хороший учебник по XML Schema, обращайтесь к «XML Schema Primer» по адресу *http://www.w3.org/TR/xmlschema-0*.

RelaxNG

RelaxNG – простой язык схем, разработанный в качестве альтернативы для XML Schema. Важное различие между ними заключается в том, что RelaxNG обходится без многих определений типов данных XML Schema. В RelaxNG документ XML проверяется по определениям типов данных, импортированным из других источников (в том числе и из XML Schema, например). Домашняя страница комитета OASIS RelaxNG находится по адресу *http://www.oasis-open.org/committees/relax-ng/*. Здесь можно найти как последнюю версию спецификации, так и учебник.

Schematron

Schematron – элегантный механизм проверки действительности документов. Он имеет простой синтаксис (всего шесть элементов), а XPath

используется для определения шаблонов в документах XML. Самая интересная и популярная реализация Schematron написана на XSLT. За дополнительной информацией (в том числе и за ссылкой на последнюю версию стандарта ISO для Schematron) обращайтесь на сайт *http://www.schematron.com/*.

The Simple API for XML (SAX)

SAX API определяет события и интерфейсы для взаимодействия с парсером SAX. SAX и DOM – самые популярные API для работы с документами XML. Полная спецификация представлена на сайте *http://www.saxproject.org/*.

Document Object Model (DOM)

Как упоминалось ранее, DOM представляет собой программный API для работы с документами. Он определяет набор интерфейсов и методов для работы с документом XML как с древовидной структурой. XSLT и XPath используют сходное древовидное представление документов XML. Домашняя страница DOM находится по адресу *http://www.w3.org/DOM/*. Страница содержит ссылки на все рекомендации W3C (уровней 1, 2 и 3) и сопутствующие документы. DOM не имеет прямого отношения к нашей теме, но хотя бы беглое знакомство с этой моделью будет полезно. (Модель данных XPath сходна с моделью DOM.)

Namespaces in XML

Как упоминалось ранее, пространства имен предотвращают конфликты имен при наличии одноименных элементов XML. Спецификация версии 1.0 размещается по адресу *http://www.w3.org/TR/REC-xml-names/*; версия 1.1 доступна по адресу *http://www.w3.org/TR/REC-xml-names11/*.

Associating stylesheets with XML documents

Документ XML может ссылаться на таблицу стилей XSLT. В этой спецификации инструкции по обработке применяются для определения одной или нескольких таблиц стилей, используемых при преобразовании документа XML. Для разных броузеров можно определить разные таблицы стилей. За полной информацией обращайтесь по адресу *http://www.w3.org/TR/xml-stylesheet*. Вот как выглядит начало документа XML с двумя связанными таблицами стилей:

```
<?xml version="1.0"?>
<?xml-stylesheet href="docbook/html/docbook.xsl" type="text/xsl"?>
<?xml-stylesheet href="docbook/wap/docbook.xsl"  type="text/xsl" media="wap"
?>
```

В этом примере первая таблица стилей используется по умолчанию, потому что она не имеет атрибута `media`. Вторая таблица используется в том случае, если поле `User-Agent` из заголовка HTTP содержит строку

wap, которая сообщает, что документ запрашивается WAP-броузером. Преимущество такого решения заключается в том, что в документе можно определить несколько разных таблиц стилей, генерирующих полезные результаты для разных броузеров или типов клиентов. Впрочем, есть и недостаток: фактически в документ XML включаются инструкции по визуализации, а этого делать не рекомендуется.

Scalable Vector Graphics (SVG)

Спецификация SVG определяет номенклатуру XML для векторной графики; иногда ее называют «PostScript с угловыми скобками». SVG позволяет определять изображения, масштабируемые до любого размера или разрешения. За подробностями обращайтесь по адресу *http://www.w3.org/TR/SVG/*.

XML pointer language (XPointer) version 1.0

XPointer предоставляет средства идентификации фрагментов веб-ресурсов. Идентификация осуществляется средствами XPath. Определение XPointer Framework находится по адресу *http://www.w3.org/TR/xptr-framework/*.

XML linking language (XLink) version 1.0

XLink определяет номенклатуру XML для ссылок на другие веб-ресурсы в документах XML. Поддерживаются как односторонние ссылки, знакомые нам всем по HTML, так и более изощренные разновидности ссылок. См. *http://www.w3.org/TR/xlink/*.

Установка процессоров XSLT

Прежде чем браться за создание таблиц стилей, посмотрим, как установить четыре популярных процессора XSLT.

Установка Xalan

В этом разделе описано, как установить процессор XSLT Xalan. В следующей главе мы создадим свою первую таблицу стилей и используем ее для преобразования документа XML.

Процесс установки относительно несложен. Предполагается, что среда JRE (Java Runtime Environment) уже установлена на вашем компьютере. Хотя Java очень редко встречается в примерах книги, сам процессор XSLT Xalan написан на Java. Установив JRE, откройте страницу *http://xml.apache.org/xalan-j/* и загрузите последнюю стабильную версию кода. (Если захочется приключений, загрузите последнюю вечернюю сборку.)

Загрузив файл Xalan в формате *.zip* или *.gzip*, распакуйте его и включите в CLASSPATH три файла: два файла *.jar* для парсера Xerces и файл

.jar самого ядра таблиц стилей Xalan. На момент написания книги эти файлы назывались *xalan.jar*, *xercesImpl.jar* и *xml-apis.jar*. (Имеется также четвертый файл *bsf.jar*, включающий Bean Scripting Framework, но мы используем его только для расширений.)

Чтобы убедиться в правильности установки Xalan, откройте командную строку и введите следующую команду:

```
java org.apache.xalan.xslt.Process
```

В строке указывается имя класса Java, поэтому регистр символов должен точно соблюдаться. Вы увидите сообщение об ошибке следующего вида:

```
java org.apache.xalan.xslt.Process
=xslproc options:
    -IN inputXMLURL
   [-XSL XSLTransformationURL]
   [-OUT outputURL]
   [-LXCIN compiledStylesheetFileNameIn]
   [-LXCOUT compiledStylesheetFileNameOutOut]
...
```

Если вы получили это сообщение, значит, все прошло успешно! Можете переходить к следующей главе, в которой мы построим свою первую таблицу стилей XSLT.

Установка Saxon

На момент написания книги наиболее полным процессором таблиц стилей XSLT 2.0, распространяемым с открытым кодом, был процессор Saxon. Он написан Майклом Кэем (Michael Kay), редактором спецификации XSLT 2.0, и доступен по адресу *http://saxon.sourceforge.net*. Загрузите файл (в настоящее время *saxonb9-0-0-2j.zip*) и включите *saxon9.jar* в `CLASSPATH`. Также архив содержит девять других файлов: *saxon9-ant.jar*, *saxon9-dom.jar*, *saxon9-dom4j.jar*, *saxon9-jdom.jar*, *saxon9-s9api.jar*, *saxon9-sql.jar*, *saxon9-xom.jar*, *saxon9-xpath.jar* и *saxon9-xqj.jar*. Файлы *.jar* обеспечивают дополнительные функции процессора; за более подробной информацией о них обращайтесь к документации Saxon. Для большинства примеров, представленных в книге, необходим только файл *saxon9.jar*.

После установки Saxon и обновления `CLASSPATH` откройте командную строку и введите следующую команду:

```
java net.sf.saxon.Transform
```

Появляется сообщение следующего вида:

```
No source file name
Saxon 9.0.0.3J from Saxonica
Usage: see http://www.saxonica.com/documentation/using-xsl/commandline.html
Options:
```

```
  -a                    Use xml-stylesheet PI, not style-doc argument
  -c:filename           Use compiled stylesheet from file
  -cr:classname         Use collection URI resolver class
  -dtd:on|off           Validate using DTD
  -expand:on|off        Expand defaults defined in schema/DTD
  -explain[:filename]   Display compiled expression tree
  -ext:on|off           Allow|Disallow external Java functions
  -im:modename          Initial mode
  -it:template          Initial template
  -l:on|off             Line numbering for source document
  -m:classname          Use message receiver class
  -o:filename           Output file or directory
  -or:classname         Use OutputURIResolver class
  -outval:recover|fatal Handling of validation errors on result document
  -p:on|off             Recognize URI query parameters
  -r:classname          Use URIResolver class
  -repeat:N             Repeat N times for performance measurement
  -s:filename           Initial source document
  -sa                   Schema-aware transformation
...
```

В сообщении об ошибке перечислены десятки всевозможных параметров. В большинстве случаев при вызове достаточно задать исходный файл XML и таблицу стилей XSLT.

Saxon также существует в версии с закрытым исходным кодом Saxon-SA, обеспечивающей полную поддержку функций XML Schema, определенных в XSLT 2.0, XPath 2.0 и XQuery 1.0. Все примеры книги, использующие схемосовместимые функции, были протестированы на коммерческой версии Saxon с закрытым исходным кодом.

Для установки схемосовместимой версии Saxon необходимо включить в `CLASSPATH` файл *saxon9sa.jar*. Приобретая Saxon-SA, вы получаете файл *saxon-license.lic*; поместите его в каталог *Saxon/bin* и включите этот каталог в системную переменную `PATH`. Команда запуска схемосовместимой версии Saxon выглядит немного иначе:

```
java com.saxonica.Transform
```

Номер версии изменяется, но все остальное остается прежним:

```
No source file name
Saxon-SA 9.0.0.3J from Saxonica
Usage: see http://www.saxonica.com/documentation/using-xsl/commandline.html
Options:
  -a                    Use xml-stylesheet PI, not style-doc argument
...
```

Установка Microsoft XSLT Processor

Процессором XSLT, чаще всего используемым в мире .NET, является Microsoft XSLT Processor. За информацией об этом инструментарии обращайтесь на страницу *http://msdn.microsoft.com/xml*. На момент

написания книги (2008 год) загружать следовало файл *msxsl.exe*. Включите его в системный путь, откройте приглашение командной строки и введите следующую команду:

```
msxsl
```

Появляется сообщение следующего вида:

```
Microsoft (R) XSLT Processor Version 4.0

Usage: MSXSL source stylesheet [options] [param=value...]
[xmlns:prefix=uri...]

Options:
    -?            Show this message
    -o filename   Write output to named file
    -m startMode  Start the transform in this mode
    -xw           Strip non-significant whitespace from source and stylesheet
...
```

В главе 9 мы рассмотрим код C#, в котором используется процессор XSLT, встроенный в инфраструктуру .NET. Если ваши потребности ограничиваются преобразованием документов XML из командной строки, *msxsl.exe* будет вполне достаточно.

Установка ядра XSLT Altova

На момент написания книги (начало 2008 года) существовал только один некоммерческий процессор XSLT 2.0 с поддержкой Schema – ядро XSLT Altova. Этот процессор XSLT заложен в основу продукта Altova XMLSpy. Сейчас загружаемый файл, работающий только в системе Windows, доступен на условиях безвозмездной лицензии по адресу *http://www.altova.com/altovaxml*.

Чтобы установить ядро, загрузите пакет (в настоящее время установочный файл называется *altovaxml2008.exe*) и запустите его. Введите в командной строке следующую команду:

```
altovaxml
```

Выводится сообщение следующего вида:

```
AltovaXML Version 2008 sp1
Copyright (c) 1998-2007 Altova GmbH. All rights reserved.
Use of this software is subject to the license agreement at
http://www.altova.com/altovaxmldla.html

Use the xslt1 engine:
   /xslt1 <filename> /in <filename> [/param name=value] [/out <filename>]
Use the xslt2 engine:
   /xslt2 <filename> /in <filename> [/param name=value] [/out <filename>]
Use the xquery engine:
   /xquery <filename> [/in <filename>] [/param name=value] [/out <filename>]
                      [serialization options]
```

```
Use the validator:
   /validate <filename> [/schema <filename> | /dtd <filename>]
   /wellformed <filename>

Parameters:
   /validate, /v <filename>   Schema validates the specified XML file
   /wellformed, /w <filename> Check if specified XML file is well-formed
   /xslt1 <filename>          Sets the source for XSLT1 stylesheet
   /xslt2 <filename>          Sets the source for XSLT2 stylesheet
   /xquery, /xq <filename>    Sets the source for XQuery expression
   /in <filename>             Sets the source for XML data
   /out <filename>            Sets the output file name
                              If omitted, data is written to stdout
   /param <name>=<value>      Adds external paramter
```

Итоги

В этой главе мы рассмотрели основы XML, а также познакомились с DOM и SAX – двумя стандартами, чаще всего используемыми процессорами XSLT. Кроме того, были рассмотрены другие технологические стандарты и способы установки некоторых процессоров таблиц стилей. К этому моменту читатель располагает всей необходимой информацией для построения и использования первых таблиц стилей, чем мы и займемся в следующей главе.

2

Обязательный пример «Hello World»

В будущих главах мы во всех подробностях рассмотрим XSLT, XPath и различные специальные функции, используемые для преобразования документов XML. Но сначала давайте разберем короткий пример, который показывает, как работают таблицы стилей.

Изучив материал этой главы, вы будете знать:

- Как создать несложную таблицу стилей
- Как использовать таблицу стилей для преобразования документа XML
- Как процессор XSLT использует таблицу стилей для преобразования документа XML
- Структуру таблицы стилей XSLT

Преобразование Hello World

В продолжение традиции примеров «Hello World», начатой еще Брайаном Керниганом (Brian Kernighan) и Деннисом Ричи (Dennis Ritchie) в книге «The C Programming Language» (Prentice Hall, 1988), мы преобразуем XML-документ «Hello World».

Пример документа

Начнем с рассмотрения исходного документа, использованного в нашем примере. Перед вами простой документ XML на базе спецификации XML 1.0, который выводит знаменитое приветствие:

```
<?xml version="1.0"?>
<!-- greeting.xml -->
<greeting>
  Hello, World!
```

```
</greeting>
```

Нам хотелось бы преобразовать этот очаровательный документ в форму, пригодную для просмотра в обычном броузере.

Пример таблицы стилей

А вот как выглядит таблица XSLT, определяющая преобразование документа XML:

```
<?xml version="1.0"?>
<!-- greeting.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <xsl:apply-templates select="greeting"/>
  </xsl:template>

  <xsl:template match="greeting">
    <html>
      <body>
        <h1>
          <xsl:value-of select="."/>
        </h1>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Вскоре мы обсудим все эти элементы и их смысл. Помните, что таблица стилей также является документом XML, поэтому в ней должны соблюдаться все правила, упоминавшиеся в предыдущей главе.

Преобразование документа XML

Чтобы преобразовать документ XML с помощью таблицы стилей XSLT, выполните следующую команду, если используете процессор Xalan:

```
java org.apache.xalan.xslt.Process -in greeting.xml -xsl greeting.xsl
  -out greeting.html
```

Для Saxon команда выглядит так:

```
java net.sf.saxon.Transform -o greeting.html greeting.xml greeting.xsl
```

Если вы используете схемосовместимую версию Saxon, имя класса Java должно быть другим:

```
java com.saxon.Transform -o greeting.html greeting.xml greeting.xsl
```

Для ядра XLST Altova применяется такая команда:

```
altovaxml /xslt1 greeting.xsl /in greeting.xml /out greeting.html
```

Наконец, если вы используете Microsoft MSXSL, введите команду:

```
msxsl greeting.xml greeting.xsl -o greeting.html
```

Команда преобразует документ *greeting.xml* по шаблонам, содержащимся в таблице стилей *greeting.xsl*. Результаты преобразования записываются в файл *greeting.html*. Откройте выходной файл в броузере и убедитесь в том, что преобразование прошло успешно.

Это одно из немногих мест книги, где мы рассматриваем синтаксис команд преобразования. Обычно бывает достаточно знать имена файлов XML, XSL и выходного файла, а также формат команды процессора XSLT. Исключения из правила встречаются при необходимости выполнения нетривиальных операций (например, передачи параметров таблицам стилей).

Результаты

Процессор XSLT генерирует следующий результат:

```
<html>
<body>
<h1>
  Hello, World!
</h1>
</body>
</html>
```

При просмотре в броузере выходной документ выглядит так, как показано на рис. 2.1.

Рис. 2.1. *HTML-версия нашего файла Hello World*

Поздравляю! Вы только что успешно использовали таблицу XSLT для преобразования документа XML.

Как обрабатываются таблицы стилей

Пока вы все еще испытываете эйфорию от преобразования документа XML, давайте разберемся, что такое таблица стилей и как она работает. В ходе освоения XSLT программист тратит немало усилий, пытаясь разобраться, как обрабатываются таблицы стилей. Чтобы упростить эту задачу, мы рассмотрим все действия, выполняемые процессором XSLT при создании нашего документа HTML.

Разбор таблицы стилей

Прежде чем процессор XSLT сможет приступить к обработке таблицы, он должен сначала прочитать ее. На концептуальном уровне неважно, как процессор XSLT хранит информацию из таблицы. Будем просто считать, что процессор XSLT может неким волшебным образом найти в таблице все, что потребуется. (Если вам так уж необходимо знать, Xalan использует оптимизированную табличную структуру для представления таблицы стилей; другие процессоры могут использовать этот способ или какой-то другой.)

Наша таблица стилей содержит три элемента: `<xsl:output>` выбирает HTML как выходной формат, а два элемента `<xsl:template>` указывают, как должны преобразовываться части документа XML.

Разбор преобразуемого документа

После обработки таблицы стилей процессор XSLT читает документ, который он должен преобразовать. Процессор XSLT строит древовидное представление на основе исходного кода XML. При построении таблицы стилей мы ориентируемся именно на это дерево.

Повторяем «до упора»

Наконец, все готово к непосредственному преобразованию документа XML. Процессор XSLT может задать некоторые свойства на основании таблицы стилей (в предыдущем примере он выбирает выходной формат HTML), после чего переходит к обработке. При этом он использует следующую логику:

1. Остались ли узлы для обработки? Обрабатываемые узлы представлены *контекстом* (*context*). Изначально контекст соответствует корню документа XML, но он перемещается по таблице в ходе обработки. Контекст подробно рассматривается в следующей главе.

Пока в контексте остаются узлы, выполняются следующие действия:

2. Получить следующий узел из контекста. Имеется ли элемент `<xsl:template>`, соответствующий ему? (В нашем примере следующим узлом является корневой узел, представленный в синтаксисе XPath обозначением /.) В таблице имеется шаблон, соответствующий этому узлу, – он начинается с `<xsl:template match="/">`.
3. Если соответствие обнаруживается в нескольких элементах `<xsl:template>`, выбрать подходящий шаблон и обработать его. «Подходящим» считается самый конкретный шаблон. Например, шаблон `<xsl:template match="/html/body/h1/p">` является более конкретным, чем `<xsl:template match="p">`. (За дополнительной информацией обращайтесь к описанию `<xsl:template>` в приложении А.) Если среди `<xsl:template>` соответствий не находится, процессор XSLT использует встроенные правила. За дополнительной инфор-

мацией обращайтесь к разделу «Встроенные правила шаблонов» далее в этой главе.

Обратите внимание: модель обработки имеет рекурсивную природу. Мы обрабатываем текущий узел, находя для него подходящий элемент `xsl:template`. В свою очередь, `xsl:template` может активизировать другие шаблоны `xsl:template`, которые тоже активизируют еще какие-нибудь `xsl:template`. К этой модели необходимо привыкнуть, но зато потом вы оцените ее элегантность.

Корневой шаблон (`<xsl:template match="/">`) может рассматриваться как аналог метода `main` в программе на C, C++ или Java. Сколько бы кода ни содержала программа, ее выполнение начинается с метода `main`. Здесь действует тот же принцип: сколько бы шаблонов `<xsl:template>` вы ни определили в своей таблице стилей, все начинается с `<xsl:template match="/">`.

Подробный анализ примера

Давайте вернемся к нашему примеру и посмотрим, как процессор XSLT преобразует документ.

1. Таблица стилей XSLT разбирается и преобразуется в древовидную структуру.
2. Документ XML также разбирается и преобразуется в древовидную структуру. Не беспокойтесь о том, как выглядит это дерево и как оно работает; пока просто считайте, что процессор XSLT знает все содержимое документа XML и таблицы стилей XSLT. После завершения первых двух шагов процессор располагает полной информацией о тех данных, с которыми мы работаем.
3. Процессор XSLT находится в корне документа XML (начальный контекст).
4. Существует шаблон `xsl:template`, соответствующий корню документа:

```
<xsl:template match="/">
  <xsl:apply-templates select="greeting"/>
</xsl:template>
```

   Косая черта (/) представляет собой *шаблон* XSLT (написанный на *XPath*). Процессор использует шаблон для поиска подходящих «узлов документа».
5. Теперь обработка переходит в `xsl:template`. Здесь содержится единственная инструкция: применить все шаблоны `xsl:template`, которые могут быть применены, ко всем элементам `greeting` в текущем контексте. Текущий контекст в шаблоне определяется атрибутом `match` элемента `xsl:template`. Это означает, что процессор XSLT должен найти все элементы `greeting` в корневом узле документа.

Так как корень документа содержит один элемент greeting, процессор XSLT должен обработать этот элемент. (Если в текущем контексте будут найдены совпадения сразу для нескольких элементов, процессор XSLT обрабатывает их последовательно в соответствии с очередностью их следования в документе; этот порядок называется *порядком документа*.) Обратившись к элементу greeting, мы видим, что к нему применим второй шаблон xsl:template нашей таблицы стилей:

```
<xsl:template match="greeting">
  <html>
    <body>
      <h1>
        <xsl:value-of select="."/>
      </h1>
    </body>
  </html>
</xsl:template>
```

6. Управление переходит к шаблону xsl:template для элемента greeting. Первые три столбца этого шаблона (<html>, <body> и <h1>) содержат элементы HTML. Так как они не определяются в объявлении пространства имен, процессор XSLT передает эти элементы HTML в выходной поток в неизменном виде.

   Средняя часть шаблона xsl:template содержит элемент xsl:value-of, который записывает некое значение в выходной поток. В данном случае используется выражение XPath «.» (точка), обозначающее текущий узел. Процессор XSLT берет текущий узел (элемент greeting, обрабатываемый в данный момент) и выводит его текст.

   Так как таблица стилей является документом XML (я не слишком часто это повторяю?), мы должны завершить элементы <h1>, <body> и <html>. На этом обработка текущего шаблона завершается, а управление возвращается шаблону, из которого он был активизирован.

7. Мы снова находимся в шаблоне корневого элемента. Все элементы <greeting> обработаны, так что выполнение этого шаблона завершено.

8. Текущий контекст не содержит других элементов (корневой элемент может быть только один). Работа процессора XSLT завершена.

Структура таблицы стилей

В завершающей части введения в XSLT описано содержимое самой таблицы стилей. Мы объясним все ее составляющие, а также обсудим другие возможные подходы.

Элемент <xsl:stylesheet>

Элемент `<xsl:stylesheet>` обычно является корневым элементом таблицы стилей XSLT:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Прежде всего элемент `<xsl:stylesheet>` определяет используемую версию XSLT и пространство имен `xsl`. Чтобы таблица стилей соответствовала спецификации XSLT, она всегда должна начинаться с этого элемента, записанного именно так, как показано выше. Некоторые процессоры таблиц стилей, в первую очередь Xalan, выдают предупреждающее сообщение, если элемент `<xsl:stylesheet>` не содержит эти два атрибута с указанными значениями. Во всех примерах книги таблица стилей начинается именно с этого элемента, а другие пространства имен определяются по мере необходимости.

Элемент <xsl:output>

Далее устанавливается метод вывода. В спецификации XSLT определены три метода вывода: `xml`, `html` и `text`. Мы создаем документ HTML, поэтому использовать следует метод HTML. Кроме этих трех методов, процессор XSLT имеет право определять собственные методы вывода; за информацией о других вариантах обращайтесь к документации своего процессора XSLT:[1]

```
<xsl:output method="html"/>
```

Для разных методов вывода используются разные атрибуты. Например, с методом `method="xml"` можно применять атрибуты `doctype-public` и `doctype-system` для определения публичного и системного идентификаторов, используемых в объявлении типа документа. С методом `method="xml"` или `method="html"` может использоваться атрибут `indent`, управляющий отступами в выходном документе. Все подробности приведены в описании элемента `<xsl:output>` в приложении А.

Первый элемент <xsl:template>

Наш первый шаблон совпадает с `"/"` – выражением XPath для корневого элемента документа:

```
<xsl:template match="/">
  <xsl:apply-templates select="greeting"/>
</xsl:template>
```

[1] *[2.0]* XSLT 2.0 также определяет метод вывода `xhtml`.

Шаблон <xsl:template> для элементов <greeting>

Второй элемент `<xsl:template>` обрабатывает все элементы `<greeting>` в исходном документе XML:

```
<xsl:template match="greeting">
  <html>
    <body>
      <h1>
        <xsl:value-of select="."/>
      </h1>
    </body>
  </html>
</xsl:template>
```

Встроенные шаблоны

Хотя в большинстве шаблонов, которые мы будем создавать в книге, способы преобразования различных элементов XML определяются явно, XSLT также определяет несколько встроенных шаблонов (правил), которые применяются при отсутствии каких-либо конкретных правил. Эти правила обладают более низким приоритетом по сравнению с другими шаблонами, поэтому при определении пользовательских шаблонов предпочтение всегда отдается последним. Далее перечислены встроенные шаблоны XSLT.

Встроенный шаблон для узлов документов и элементов

Этот шаблон обрабатывает узел документа и все его дочерние узлы. Он гарантирует, что рекурсивная обработка продолжится даже в том случае, если для некоторого элемента шаблон не определен:

```
<xsl:template match="*|/">
  <xsl:apply-templates/>
</xsl:template>
```

Для примера возьмем следующий документ:

```
<?xml version="1.0"?>
<x>
  <y>
    <z/>
  </y>
</x>
```

Встроенное правило для узлов документов и элементов означает, что мы можем написать таблицу стилей, содержащую единственный шаблон с `match="z"`, – и элемент `<z>` все равно будет обработан, несмотря на отсутствие шаблонов для элементов `<x>` и `<y>`.

Встроенный шаблон для режимов

Этот шаблон гарантирует, что узлы документов и элементов будут обработаны независимо от действующего режима. (За информацией об атрибуте `mode` обращайтесь к разделу «Шаблоны и атрибут mode» в главе 5.)

```
<xsl:template match="*|/" mode="x">
  <xsl:apply-templates mode="x"/>
</xsl:template>
```

Встроенный шаблон для узлов текста и атрибутов

Шаблон копирует текст из всех узлов текста и атрибутов в выходное дерево. Учтите, что для активизации этого шаблона узлы текста и атрибутов должны быть выделены:

```
<xsl:template match="text()|@*">
  <xsl:value-of select="."/>
</xsl:template>
```

Встроенный шаблон для комментариев и инструкций по обработке

Этот шаблон не делает ничего:

```
<xsl:template match="comment()|processing-instruction()"/>
```

Встроенный шаблон для узлов пространств имен

Этот шаблон тоже ничего не делает:

```
<xsl:template match="namespace()"/>
```

Элементы верхнего уровня

До настоящего момента мы фактически не обсуждали ни исходный документ, ни то, как мы собираемся его преобразовывать. Мы только устанавливали некоторые свойства преобразования. Существуют специальные элементы, которые могут размещаться в начале таблицы стилей. Любой элемент, родительским элементом которого является `<xsl:stylesheet>`, называется *элементом верхнего уровня*. Краткая сводка элементов верхнего уровня:

`<xsl:include>` и `<xsl:import>`

Эти элементы указывают на другую таблицу стилей, которая вместе со всем своим содержимым включается в текущую таблицу. Главное различие между `<xsl:import>` и `<xsl:include>` состоит в том, что шаблон, переменная или то, что импортируется элементом `<xsl:import>`, обладает более низким приоритетом, чем содержимое текущей таблицы стилей. Если взглянуть на происходящее с объектно-ориентированной точки зрения, в вашем распоряжении появляется механизм субклассирования таблиц стилей. Вы можете импор-

тировать другую таблицу, содержащую общие шаблоны, но вместо шаблонов импортируемой таблицы по возможности будут использоваться шаблоны *импортирующей* таблицы. Есть и другое различие: элемент <xsl:import> может находиться только в начале таблицы стилей, тогда как <xsl:include> может располагаться в любом месте.

<xsl:strip-space> и <xsl:preserve-space>

Эти элементы могут содержать список элементов (разделенных пробелами), для которых следует удалить или сохранить пропуски в выходных данных. Чтобы определить такие элементы глобально, используйте запись <xsl:strip-space elements="*"/> или <xsl:preserve-space elements="*"/>. Если же потребуется указать, что пропуски должны быть исключены из всех элементов, кроме <greeting> и <salutation>, в таблицу стилей добавляется следующий фрагмент разметки:

```
<xsl:strip-space elements="*"/>
<xsl:preserve-space elements="greeting
            salutation"/>
```

<xsl:key>

Элемент определяет ключ – аналог индекса при поиске в базе данных. Элемент <xsl:key> и функция key() более подробно рассматриваются в разделе «Элементы программной логики в XSLT» главы 5.

<xsl:variable>

Элемент определяет переменную. Все переменные <xsl:variable>, находящиеся в элементах верхнего уровня, являются глобальными для всей таблицы стилей. Переменные подробно рассматриваются в разделе «Переменные» главы 5.

<xsl:param>

Элемент определяет параметр. Как и в случае с переменными <xsl:variable>, все параметры <xsl:param>, находящиеся в элементах верхнего уровня, являются глобальными для всей таблицы стилей. Параметры подробно рассматриваются в разделе «Параметры» главы 5.

Другие элементы

На верхнем уровне также могут находиться другие реже используемые элементы: <xsl:decimal-format>, <xsl:namespace-alias> и <xsl:attribute-set>. Они описаны в приложении А.

Другие подходы

Главная мантра сообщества Perl гласит: «Это можно сделать несколькими способами». Принцип справедлив и для таблиц стилей XSLT. Например, нашу таблицу можно записать следующим образом:

```
<?xml version="1.0"?>
<!-- greeting2.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <body>
        <xsl:apply-templates select="greeting"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="greeting">
    <h1>
      <xsl:value-of select="."/>
    </h1>
  </xsl:template>
</xsl:stylesheet>
```

В этой версии элементы-«обертки» документа HTML включены в шаблон корневого элемента. В ходе проектирования собственных таблиц стилей вы должны подумать над тем, где следует размещать такие элементы, как `<html>` и `<body>`. Предположим, документ XML выглядит так:

```
<?xml version="1.0"?>
<greetings>
  <greeting>Hello, World!</greeting>
  <greeting>Hey, Y'all!</greeting>
</greetings>
```

В этом случае элементы `<html>` и `<body>` должны размещаться в шаблоне `<xsl:template>` корневого элемента. Если включить их в `<xsl:template>` элемента `<greeting>`, то выходной документ будет содержать несколько элементов `<html>`, что недопустимо в документах HTML. Обновленная таблица стилей может выглядеть так:

```
<?xml version="1.0"?>
<!-- multiple-greetings.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <body>
        <xsl:apply-templates select="greetings/greeting"/>
      </body>
    </html>
  </xsl:template>
```

```
  <xsl:template match="greeting">
    <h1>
      <xsl:value-of select="."/>
    </h1>
  </xsl:template>
</xsl:stylesheet>
```

Обратите внимание на измененное выражение XPath: исходное имя `greeting` превратилось в `greetings/greeting`. В ходе разработки таблиц стилей необходимо следить за тем, чтобы выражения XPath соответствовали структуре документа. Если вы получаете неожиданные результаты (или не получаете никаких результатов), как правило, виновато неправильное выражение XPath.

И последний пример: нашу таблицу стилей также можно записать с единственным шаблоном `xsl:template`:

```
<?xml version="1.0"?>
<!-- greeting3.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <body>
        <h1>
          <xsl:value-of select="greeting"/>
        </h1>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Несмотря на то что эта таблица получилась самой короткой, в своих примерах мы обычно будем использовать несколько коротких шаблонов, каждый из которых определяет простое преобразование для нескольких элементов. Такой подход значительно упрощает понимание, сопровождение и повторное использование таблиц стилей. Чем больше преобразований вы втиснете в каждый шаблон, тем труднее отладить таблицу стилей и заново использовать шаблоны в другой ситуации.

Галерея преобразований

Прежде чем переходить к более сложным темам, мы рассмотрим еще несколько преобразований нашего документа Hello World. Простые таблицы стилей, представленные в этом разделе, преобразуют документ XML в следующие форматы:

- Файл SVG (Scalar Vector Graphics)
- Файл PDF

- Программа Java
- Файл VRML (Virtual Reality Modeling Language)

Файл Hello World в формате SVG

В первом примере документ Hello World преобразуется в файл SVG:

```
<?xml version="1.0"?>
<!-- svg-greeting.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg">

  <xsl:template match="/">
    <svg:svg width="10cm" height="4cm">
      <svg:g>
        <svg:defs>
          <svg:radialGradient id="MyGradient"
            cx="4cm" cy="2cm" r="3cm" fx="4cm" fy="2cm">
            <svg:stop offset="0%" style="stop-color:red"/>
            <svg:stop offset="50%" style="stop-color:blue"/>
            <svg:stop offset="100%" style="stop-color:red"/>
          </svg:radialGradient>
        </svg:defs>
        <svg:rect style="fill:url(#MyGradient); stroke:black"
          x="1cm" y="1cm" width="8cm" height="2cm"/>
        <svg:text x="5.05cm" y="2.25cm" text-anchor="middle"
          style="font-family:Verdana; font-size:24;
          font-weight:bold; fill:black">
          <xsl:apply-templates select="greeting"/>
        </svg:text>
        <svg:text x="5cm" y="2.2cm" text-anchor="middle"
          style="font-family:Verdana; font-size:24;
          font-weight:bold; fill:white">
          <xsl:apply-templates select="greeting"/>
        </svg:text>
      </svg:g>
    </svg:svg>
  </xsl:template>
  <xsl:template match="greeting">
    <xsl:value-of select="."/>
  </xsl:template>

</xsl:stylesheet>
```

Как видно из таблицы стилей, большая часть кода всего лишь определяет структуру документа SVG. Это характерно для многих таблиц стилей; зная выходной формат, вы просто извлекаете содержимое из исходного документа XML и вставляете его в правильную позицию выходного документа. Преобразование документа Hello World по приведенной таблице стилей дает следующий результат:

```
<?xml version="1.0" encoding="UTF-8"?>
<svg:svg xmlns:svg="http://www.w3.org/2000/svg" width="10cm" height="4cm">
  <svg:g>
    <svg:defs>
      <svg:radialGradient id="MyGradient" cx="4cm" cy="2cm" r="3cm"
           fx="4cm" fy="2cm">
        <svg:stop offset="0%" style="stop-color:red"/>
        <svg:stop offset="50%" style="stop-color:blue"/>
        <svg:stop offset="100%" style="stop-color:red"/>
      </svg:radialGradient>
    </svg:defs>
    <svg:rect style="fill:url(#MyGradient); stroke:black" x="1cm"
      y="1cm" width="8cm" height="2cm"/>
    <svg:text x="5.05cm" y="2.25cm" text-anchor="middle"
      style="font-family:Verdana; font-size:24; font-weight:bold; fill:black"
>
      Hello, World!
    </svg:text>
    <svg:text x="5cm" y="2.2cm" text-anchor="middle"
      style="font-family:Verdana; font-size:24; font-weight:bold; fill:white"
>
      Hello, World!
    </svg:text>
  </svg:g>
</svg:svg>
```

На рис. 2.2 показано, как выглядит документ в программе просмотра графики SVG.

Иллюстрация была сделана в подключаемом модуле Adobe SVG для броузера Internet Explorer. Модуль можно загрузить по адресу *http://www.adobe.com/svg/*. (Многие современные броузеры обладают встроенной поддержкой SVG.)

***Рис. 2.2.** SVG-версия файла Hello World*

Файл Hello World в формате PDF

Преобразование файла Hello World в формат PDF начинается с преобразования файла XML в объекты форматирования. Способ визуализации содержимого описывается номенклатурой XML, которая называется XSL-FO (The Extensible Stylesheet Language for Formatting Objects). Таблица выглядит так:

```
<?xml version="1.0"?>
<!-- fo-greeting.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">

  <xsl:output method="xml"/>

  <xsl:template match="/">
    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
      <fo:layout-master-set>
        <fo:simple-page-master master-name="standard"
          margin-right="75pt" margin-left="75pt"
          page-height="11in" page-width="8.5in"
          margin-bottom="25pt" margin-top="25pt">
          <fo:region-body margin-top="50pt" margin-bottom="50pt"/>
        </fo:simple-page-master>
      </fo:layout-master-set>

      <fo:page-sequence master-reference="standard">
        <fo:flow flow-name="xsl-region-body">
          <xsl:apply-templates select="greeting"/>
        </fo:flow>
      </fo:page-sequence>
    </fo:root>
  </xsl:template>

  <xsl:template match="greeting">
    <fo:block line-height="76pt" font-size="72pt" text-align="center">
      <xsl:value-of select="."/>
    </fo:block>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей преобразует документ Hello World в следующий файл XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="standard" margin-right="75pt"
      margin-left="75pt" page-height="11in" page-width="8.5in"
      margin-bottom="25pt" margin-top="25pt">
      <fo:region-body margin-top="50pt" margin-bottom="50pt"/>
    </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-reference="standard">
    <fo:flow flow-name="xsl-region-body">
      <fo:block line-height="76pt" font-size="72pt" text-align="center">
        Hello, World!
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
```

```
</fo:root>
```

В этом длинном наборе тегов используются объекты форматирования, описывающие размер страницы, величину полей, размеры шрифтов, высоту строк и т. д., а также текст, извлеченный из исходного документа XML. После получения объектов форматирования остается воспользоваться утилитой FOP из Apache XML Project. На рис. 2.3 показан файл PDF, полученный в результате преобразования объектов форматирования в PDF.

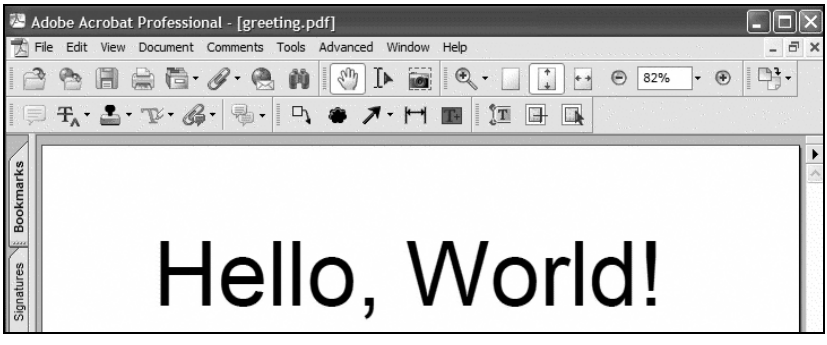


Рис. 2.3. *PDF-версия файла Hello World*

Команда преобразования файла с объектами форматирования в формат PDF выглядит так:

```
java org.apache.fop.apps.Fop greeting.fo greeting.pdf
```

Программа Hello World на Java

В двух последних преобразованиях номенклатуры XML вообще не используются; преобразование документа Hello World в другие форматы осуществляется средствами XSLT. Далее мы трансформируем документ XML в исходный код программы Java. После компиляции и выполнения программа выводит сообщение из документа XML на консоль. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- java-greeting.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text" encoding="UTF-8"/>

  <xsl:template match="/">
    <xsl:text>
public class Greeting
{
  public static void main(String[] argv)
  {
```

```
    </xsl:text>
    <xsl:apply-templates select="greeting"/>
    <xsl:text>
  }
}
    </xsl:text>
  </xsl:template>

  <xsl:template match="greeting">
    <xsl:text>System.out.println("</xsl:text>
    <xsl:value-of select="normalize-space()"/>
    <xsl:text>");</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

(Обратите внимание: элемент `<xsl:output method="text">` выбирает метод построения текста, а не разметки.) Для этой таблицы стилей создается следующий результат:

```
public class Greeting
{
  public static void main(String[] argv)
  {
    System.out.println("Hello, World!");
  }
}
```

Имя класса, определяемое в таблице стилей XSLT (`Greeting`), должно совпадать с именем сгенерированного файла. Это означает, что при выполнении преобразования необходимо задать имя файла с точным соблюдением регистра символов. Вот как это делается для Xalan:

```
java org.apache.xalan.xslt.Process -in greeting.xml -xsl java-greeting.xsl
    -out Greeting.java
```

Для Saxon синтаксис получается чуть проще:

```
java net.sf.saxon.Transform -o Greeting.java greeting.xml java-greeting.xsl
```

Как и прежде, команда для схемосовместимой версии Saxon несколько отличается:

```
java.com.saxonica.Transform -o Greeting.java greeting.xml java-greeting.xsl
```

Наконец, для MSXSL и ядра XSLT Altova используются следующие команды:

```
msxsl -o Greeting.java greeting.xml java-greeting.xsl
```

и

```
altovaxml /xslt1 java-greeting.xsl /in greeting.xml /out Greeting.java
```

Результат выполнения сгенерированной программы Java выглядит так:

```
C:\> java Greeting
Hello, World!
```

Возможно, кому-то идея построения кода Java на основе документа XML покажется странной, но эта технология весьма полезна. В частности, программа FOP из проекта Apache XML Project определяет ряд свойств в XML, а затем генерирует исходный код Java для создания определений классов и методов `get/set` для каждого из этих свойств.

Программа Hello World в формате VRML

В последнем примере преобразования мы построим файл VRML на основе исходного документа XML. Таблица стилей для решения этой задачи выглядит так:

```
<?xml version="1.0"?>
<!-- vrml-greeting.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>#VRML V2.0 utf8

Shape
{
  geometry ElevationGrid
  {
    xDimension 9
    zDimension 9
    xSpacing 1
    zSpacing 1
    height
    [
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0 0
    ]
    colorPerVertex FALSE
    color Color
    {
      color
      [
        0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1,
        1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0,
        0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1,
```

```
        1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0,
        0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1,
        1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0,
        0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1,
        1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0,
      ]
    }
  }
}

Transform
{
  translation 4.5 1 4
  children
  [
    Shape
    {
      geometry Text
      {
    </xsl:text>
    <xsl:apply-templates select="greeting"/>
    <xsl:text>
        fontStyle FontStyle
        {
          justify "MIDDLE"
          style "BOLD"
        }
      }
    }
  ]
}

NavigationInfo
{
  type ["EXAMINE", "ANY"]
}

Viewpoint
{
  position 4 1 10
}
    </xsl:text>
  </xsl:template>

  <xsl:template match="greeting">
    <xsl:text>string"</xsl:text>
    <xsl:value-of select="normalize-space()"/>
    <xsl:text>"</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Как и предыдущая таблица стилей, наш шаблон для построения кода VRML в основном состоит из стандартной «заготовки» кода, в которую

в нужных местах вставляется содержимое из исходного документа XML. Элемент `<xsl:apply-templates>` заменяется значением элемента `<greeting>`.

Код VRML рисует шахматный узор, а затем выводит текст из документа XML, «висящий в воздухе» в центре документа. Пара изображений из VRML-версии документа XML показана на рис. 2.4 и 2.5.

Иллюстрации были сделаны в программе просмотра VRML Cortona (*http://www.parallelgraphics.com/cortona*).

Рис. 2.4. *Одно из состояний VRML-версии документа Hello World*

Рис. 2.5. *Другое состояние VRML-версии документа Hello World*

Хотя мы еще не обсуждали конкретные номенклатуры для файловых форматов, которые использовались в этих примерах, надеюсь, вы поняли, что документ XML можно преобразовать в любой полезный формат по вашему усмотрению. В оставшейся части книги мы рассмотрим несколько стандартных задач, часто решаемых средствами XSLT; все они основаны на материале этой главы.

Итоги

Приведенные таблицы стилей были достаточно тривиальны, но при этом они гораздо проще соответствующего процедурного кода (написанного на Visual Basic, C++, Java и т. д.), обеспечивающего сходные преобразования. В этой главе мы рассмотрели основы строения таблиц стилей и их работы.

В дальнейших главах мы продемонстрируем невероятно широкий спектр возможностей таблиц XSLT, в том числе:

- Использование программной логики и передачи управления
- Сортировка и группировка элементов
- Связывание и создание перекрестных ссылок между элементами
- Создание «главных документов», в которые встраиваются другие документы XML, с последующей сортировкой, фильтрацией, группировкой и форматированием объединенных документов
- Добавление новых функций процессора XSLT посредством механизма расширения XSLT

На базе технологии XSLT сформировалось чрезвычайно активное сообщество пользователей. Чтобы познакомиться с ним поближе, посетите сайт XSL-List по адресу *http://www.mulberrytech.com/xsl/xsl-list/index.html*.

Но прежде чем браться за эти темы, мы должны сначала познакомиться с XPath – синтаксисом для описания частей документа, преобразуемых к другому формату.

3

XPath: описание иголки в стоге сена

Синтаксис XPath предназначен для описания частей документа XML. Например, XPath позволяет идентифицировать первый элемент `<para>`, атрибут `quantity` элемента `<part-number>`, все элементы `<first-name>` с текстом `"Joe"` и многие другие разновидности. В таблицах стилей шаблоны XSLT в атрибутах `match` и `select` различных элементов используют синтаксис XPath для описания способа преобразования документа. В этой главе мы рассмотрим XPath во всем его блеске.

Синтаксис XPath проектировался для использования в атрибутах документов XML. Базовые выражения языков программирования (например, `$x*6`) объединяются в нем с описаниями путей в стиле UNIX (например, `/sonnet/author/last-name`). Помимо базового синтаксиса XPath предоставляет набор полезных функций для получения разнообразной информации о документе.

Следует помнить одно важное обстоятельство: XPath работает с разобранной версией документа XML. Это означает, что некоторые подробности исходного документа в XPath недоступны. Например, ссылки на сущности преобразуются процессором XSLT до обработки инструкций стилевой таблицы. Секции CDATA также преобразуются в текст. Это означает, что мы не можем определить, чем был узел текста дерева XPath в исходном документе XML – текстом, ссылкой на сущность или частью секции CDATA. Постепенно вы привыкнете рассматривать документы XML в контексте выражений XPath, и проблемы исчезнут, но на первых порах это может вас смущать.

[2.0] В версии 2.0 спецификации XPath произошли огромные изменения. Все, что работало в XPath 1.0, по-прежнему работает и в XPath 2.0, но появились новые возможности и операторы, значительно упрощающие вашу жизнь при использовании процессора XSLT 2.0. В этой главе мы отдельно рассмотрим три основные темы XPath 2.0:

- В представлении XPath 2.0 все, с чем мы имеем дело, является *последовательностью*. Последовательность заменяет концепцию наборов узлов, использовавшуюся в XPath 1.0. Главное отличие заключается в том, что последовательность может содержать атомарные значения (подробности чуть позже). Помните, что узлы последовательности (например, узел документа) может иметь дочерние узлы. Когда мы работаем с частями документа, последовательность часто представляет собой узел элемента с дочерними узлами и атрибутами; эта последовательность из одного узла работает точно так же, как деревья, хорошо знакомые нам по XPath 1.0.
- XPath 2.0 поддерживает *атомарные значения*. В XPath 1.0 все результаты разбора документа XML, включая комментарии, текст и инструкции по обработке, представлялись узлами. Система типов была относительно простой: строки, числа, логические величины, наборы узлов и так называемые фрагменты итогового дерева. В XPath 2.0 последовательность может содержать узлы из документа XML наряду со значением 42 типа `xs:integer`, или же значение `xs:date` может стать частью последовательности вместе с узлами, использовавшимися в XPath 1.0. Атомарные узлы определяются в спецификации XML Schema; атомарным называется значение, которое не может быть разбито на меньшие части. Например, значение типа `xs:integer` или `xs:date` является атомарным.
- XPath 2.0 поддерживает все встроенные типы данных XML Schema. Это означает, что мы можем указать, что значение относится к определенному типу данных, создать новое значение определенного типа данных или же преобразовать заданное значение к конкретному типу данных. Если ваш процессор XSLT 2.0 является схемосовместимым (об этом чуть позже), вы также можете определять собственные типы данных и использовать их в XPath наряду с типами данных XML Schema. Если процессор XSLT 2.0 не является схемосовместимым, он все же поддерживает некоторые типы данных из XML Schema.

Изменения XPath 2.0 будут особо упоминаться при обсуждении XPath в целом (кроме трех указанных исключений).

Модель данных в XPath

XPath 1.0 рассматривает документ XML как дерево узлов. Это дерево очень похоже на дерево DOM (Document Object Model); если вы знакомы с DOM, то уже имеете некоторое представление о построении несложных выражений XPath. (Точнее, речь идет о концептуальном дереве; процессор XSLT или другая программа, реализующая стандарт XPath, не обязана строить «реальное» дерево.)

[2.0] В XPath 2.0 все рассматривается как последовательность. Последовательность может содержать все узлы, которые обсуждаются в этом

разделе, а также атомарные значения. При работе с разобранными данными из документа XML последовательности чаще всего соответствуют узлам из дерева документа, так что об атомарных значениях можно не беспокоиться. Пока достаточно запомнить, что в XPath 2.0 используется несколько иная базовая модель данных; различия более подробно рассматриваются далее в этой главе.

Типы узлов

В XPath существуют семь разновидностей узлов:

- Узел документа (только один в каждом документе)
- Узлы элементов
- Узлы атрибутов
- Узлы текста
- Узлы комментариев
- Узлы инструкций по обработке
- Узлы пространств имен

Мы обсудим разные типы узлов на примере следующего документа:

```
<?xml version="1.0"?>
<?xml-stylesheet href="sonnet.xsl" type="text/xsl"?>

<!DOCTYPE sonnet [
  <!ELEMENT sonnet (auth:author, title, lines)>
  <!ATTLIST sonnet public-domain CDATA "yes"
            type (Shakespearean | Petrarchan) "Shakespearean">
  <!ELEMENT auth:author  (last-name,first-name,nationality,
                          year-of-birth?,year-of-death?)>
  <!ELEMENT last-name (#PCDATA)>
  <!ELEMENT first-name (#PCDATA)>
  <!ELEMENT nationality (#PCDATA)>
  <!ELEMENT year-of-birth (#PCDATA)>
  <!ELEMENT year-of-death (#PCDATA)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT lines (line,line,line,line,
                   line,line,line,line,
                   line,line,line,line,
                   line,line)>
  <!ELEMENT line (#PCDATA)>
]>

<!-- По умолчанию используется тип сонета Shakespearean, -->
     также допустим тип "Petrarchan."                     -->
<sonnet type='Shakespearean'>
  <auth:author xmlns:auth="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
```

```
    <year-of-death>1616</year-of-death>
  </auth:author>
  <!-- Существует ли официальное название сонета?
       Иногда сонеты называются по первой строке. -->
  <title>Sonnet 130</title>
  <lines>
    <line>My mistress' eyes are nothing like the sun,</line>
    <line>Coral is far more red than her lips red.</line>
    <line>If snow be white, why then her breasts are dun,</line>
    <line>If hairs be wires, black wires grow on her head.</line>
    <line>I have seen roses damasked, red and white,</line>
    <line>But no such roses see I in her cheeks.</line>
    <line>And in some perfumes is there more delight</line>
    <line>Than in the breath that from my mistress reeks.</line>
    <line>I love to hear her speak, yet well I know</line>
    <line>That music hath a far more pleasing sound.</line>
    <line>I grant I never saw a goddess go,</line>
    <line>My mistress when she walks, treads on the ground.</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
  </lines>
</sonnet>
<!-- Название альбома Стинга 1987 "Nothing like the sun" (1987) -->
<!-- позаимствовано из строки 1 этого сонета.                    -->
```

Корневой узел

Корневым узлом является узел XPath, содержащий весь документ. В нашем примере корневой узел содержит элемент `<sonnet>`; это не сам элемент `<sonnet>`! В выражениях XPath корневой узел задается косой чертой (/).

В отличие от других узлов, у корневого узла нет родителя. Он всегда имеет хотя бы один дочерний узел: элемент документа. Корневой узел также содержит комментарии и инструкции по обработке, не входящие в элемент документа. В нашем примере инструкция `xml-stylesheet` является дочерней по отношению к корневому узлу, как и комментарии перед тегом `<sonnet>` и после него. Строковое значение корневого узла (возвращаемое `<xsl:value-of select="/" />`) представляет собой конкатенацию всех текстовых узлов потомков корневого узла без разделения пробелами.

Узлы элементов

Каждый элемент исходного документа XML представляется узлом документа XPath. В предыдущем документе узлы элементов существуют для элемента `<sonnet>`, элемента `<auth:author>`, элемента `<last-name>` и т. д. В число дочерних узлов элемента входят узлы текста, узлы элементов, узлы комментариев и узлы инструкций по обработке, находящиеся внутри этого элемента в исходном документе.

Строковое значение узла элемента (возвращаемое, например, конструкцией <xsl:value-of select="sonnet">) представляет собой результат конкатенации текста самого узла и всех его дочерних узлов в порядке документа (то есть в порядке их следования в исходном документе). Все ссылки на сущности (например, <) и ссылки на символы (например, ß – символ «длинное S» в нижнем регистре) в тексте преобразуются автоматически. XPath воспринимает эти символы как < и ß; обратиться к ссылкам сущностей или символов из XPath невозможно.

Имя узла элемента (возвращаемое функцией XPath name()) состоит из собственно имени элемента и всех действующих пространств имен. В предыдущем примере для элемента <sonnet> функция name() вернет строку sonnet, а для элемента <auth:author> – строку auth:author. В XPath имеются функции, которые по имени узла возвращают локальное имя элемента (author) и URI-адреса пространства имен, связанного с узлом (http://www.authors.com). В XPath 2.0 появились дополнительные функции для работы с полными именами (значениями XML Schema xs:QName), включая функции для извлечения префикса пространства имен и связанного с ним URI-адреса.

Узлы атрибутов

Как минимум узел элемента является родителем одного узла атрибута для каждого атрибута в исходном документе XML. В нашем примере узел элемента, соответствующий элементу <sonnet>, является родителем узла атрибута с именем type и значением Shakespearean. Однако при обработке узлов атрибутов существуют некоторые сложности:

- Хотя узел элемента является родителем своих узлов атрибутов, эти узлы атрибутов не являются дочерними по отношению к нему. Дочерними узлами элемента являются узлы текста, элементов, комментариев и инструкций по обработке, содержащихся в исходном элементе. Если вам потребуются атрибуты документа, необходимо запросить их явно. На первый взгляд такие отношения выглядят странно, но вы вскоре убедитесь, что отдельная интерпретация атрибутов элемента – именно то, что нужно.
- Если DTD или Schema определяют значения по умолчанию для некоторых атрибутов, такие атрибуты не обязаны присутствовать в документе XML. В нашем примере объявляется атрибут public-domain со значением по умолчанию yes. Реально присутствующий в документе элемент <sonnet> не имеет этого атрибута, поэтому значение атрибута public-domain равно yes. Аналогично по умолчанию type имеет значение Shakespearean, поэтому для элемента <sonnet> без атрибута type используется значение по умолчанию.

  Ситуация усугубляется тем, что парсер XML не обязан читать внешний файл DTD. Если он этого не сделает, все узлы атрибутов, представляющие значения по умолчанию и не закодированные в документе, не будут существовать. К счастью, XSLT содержит некото-

рые элементы программной логики (<xsl:if> и <xsl:choose>), которые помогают разрешать подобные неоднозначности; они будут рассмотрены в главе 5.

- В спецификации XML 1.0 определяются два атрибута (xml:lang и xml:space), которые работают иначе. Если элемент <auth:author> в нашем примере содержит атрибут xml:lang="en-US", этот атрибут применяется ко всем элементам, содержащимся в <auth:author>. Но даже если этот атрибут применяется к элементу <last-name>, <last-name> не будет иметь атрибута с именем xml:lang. Аналогичным образом работает атрибут xml:space, определяющий, будут ли сохраняться пропуски в элементе; допустимыми значениями этого атрибута являются preserve и default. Независимо от того, действуют эти атрибуты для заданного элемента или нет, узел элемента будет содержать только те узлы атрибутов, которые присутствуют в документе или имеют значение по умолчанию в DTD.

  Пропуски будут обсуждаться в разделе «Обработка пропусков» главы 4. За подробной технической информацией о кодах языков и обработке пропусков обращайтесь к описанию функции XPath lang() в приложении C, а также к описанию элементов XSLT <xsl:preserve-space> и <xsl:strip-space> в приложении A.

Узлы текста

Узлы текста[1] создают гораздо меньше сложностей; они просто содержат текст из элемента. Если исходный текст в документе XML содержит ссылки на сущности или символы, они разрешаются до создания узла текста XPath. Узлы текста содержат текст, тривиальный и простой. Они должны содержать как можно больший объем текста; следующий или предыдущий узел текстовым быть не может.

Возможно, вы заметили, что в списке отсутствуют узлы CDATA. Если документ XML содержит текст в секции CDATA, содержимое секции будет для вас доступно в виде узла текста. При этом вы не будете знать, что данный узел текста когда-то был секцией CDATA. Аналогичным образом все ссылки на сущности преобразуются до обработки таблицы стилей, поэтому вы не знаете, содержал ли заданный блок текста ссылки на сущности или нет.

Узлы комментариев

Узлы комментариев тоже очень просты – у них есть значение, но нет имени. Каждый комментарий в исходном документе (кроме комментариев в DTD) становится узлом комментария. Значение узла комментария представляет собой все, что находится внутри комментария, т. е. весь текст между открывающим маркером <!-- и закрывающим -->.

---

[1] Также будет использоваться термин «текстовые узлы». – *Примеч. перев.*

Узлы инструкций по обработке

Узел инструкции по обработке состоит из двух частей: имени (возвращаемого функцией `name()`) и строкового значения. Строковое значение включает все, что следует после имени, в том числе и пропуски, кроме маркера `?>`, закрывающего инструкцию по обработке.

Узлы пространств имен

Узлы пространств имен почти никогда не используются в таблицах XSLT; они существуют исключительно для удобства процессора XSLT. Запомните, что объявление пространства имен (например, `xmlns:auth="http://www.authors.net"`), внешне напоминающее атрибут в исходном коде XML, становится узлом пространства имен, а не узлом атрибута. Каждый элемент в области действия объявления пространства имен имеет узел пространства имен. Так, в нашем примере каждый узел в `<auth:author>` содержит узел пространства имен.

[2.0] В XPath 2.0 поддержка узлов пространств имен не обязательна.

Тесты узлов

В XPath также существуют *тесты узлов* (*node tests*), внешне напоминающие функции, но используемые для идентификации определенных типов узлов. Тесты узлов работают как предикаты, возвращающие только те узлы, которые удовлетворяют некоторому критерию. В XPath 1.0 определены четыре теста узлов:

`node()`

Соответствует всем узлам. Тест `node()` дает истинный результат для любого вида узлов.

`text()`

Соответствует только узлам текста.

`comment()`

Соответствует узлам комментариев.

`processing-instruction()`

Соответствует узлам инструкций по обработке. Если тест узлов включает строку, он соответствует узлам с инструкциями по обработке с заданным именем. Например, `processing-instruction('cocoon-process')` соответствует узлам инструкций по обработке, начинающимся с `<?cocoon-process>`.

`*`

Соответствует всем узлам конкретной оси (оси будут рассмотрены позднее). Например, запись `child::*` соответствует всем дочерним элементам узла, тогда как `attribute::*` соответствует всем атрибутам узла.

`NCName:*`

Соответствует всем узлам заданного пространства имен. В нашем примере `auth:*` соответствует всем элементам с URI-адресом пространства имен `http://www.authors.com/`.

[2.0] Новые тесты узлов в XPath 2.0

В XPath 2.0 используются все тесты узлов из XPath 1.0, а также определяется ряд новых тестов узлов:

`element()`

Соответствует любому элементу. Использование теста узлов без аргументов аналогично `select="*"` в XPath 1.0. Различие заключается в том, что в XPath 2.0 тест узлов `element()` может использоваться для проверки имени и типа данных. Например, `element(author)` соответствует любым элементам с именем `<author>`. Версия теста с двумя аргументами позволяет найти все элементы с заданным типом данных. Тест `element(date-of-birth, xs:gYear)` соответствует всем элементам `<date-of-birth>` с типом данных `xs:gYear`. (А также с типом данных, производным от `xs:gYear`.) Наконец, при использовании метасимвола вместо имени элемента тест ищет все элементы с заданным типом данных. Так, `element(*, xs:gYear)` соответствует всем элементам с типом данных `xs:gYear`.

`schema-element(author)`

Соответствует любому элементу с именем `<author>`, тип данных которого соответствует типу данных элемента `<author>`, объявленного в схеме. В отличие от других тестов узлов, определяемых в XPath 1.0 и 2.0, использование этого теста без аргумента является ошибкой.

`attribute()`

Соответствует любому атрибуту. При использовании без аргументов тест аналогичен конструкции `select="@*"` в XPath 1.0. Как и в случае с `element()`, тест `attribute()` может использоваться для проверки имени и типа данных узлов. Тест `attribute(public-domain)` соответствует любым атрибутам с именем `public-domain` независимо от типа данных. Тест `attribute(public-domain, xs:string)` соответствует атрибутам с именем `public-domain`, имеющим тип данных `xs:string`. Наконец, при замене имени *атрибута* метасимволом тест возвращает все атрибуты с указанным типом данных. Например, `attribute(*, xs:decimal)` соответствует всем атрибутам с типом данных `xs:decimal` независимо от имени атрибута.

`*:NCName`

Соответствует всем узлам с заданным локальным именем. В нашем примере тест `*:author` соответствует элементу `<auth:author>`; он также будет соответствовать элементам `<xyz:author>` и `<author>`. Соответствие определяется независимо от URI-адреса пространства имен (и даже от его наличия).

`document-node()`

Соответствует узлам документов. Тест `document-node(element(sonnet))` соответствует узлу документа с единственным дочерним элементом (`<sonnet>`). Узел документа также может содержать комментарии и инструкции по обработке, но дочерний узел элемента может быть только один.

[2.0] Хотя `item()` внешне похожа на тест узлов, она используется только как тип данных. Например, элемент `<xsl:variable name="something" as="item()">` определяет переменную, которая может содержать только одно значение. Переменная может содержать любой узел или атомарный тип.

[2.0] Последовательности и атомарные значения

Вероятно, главные изменения в XSLT 2.0 проявляются в работе с моделью данных. Изменения в модели будут упоминаться по ходу изложения, но три темы будут изложены немедленно: последовательности, атомарные значения и поддержка схем. Начнем с последовательностей и атомарных значений, а поддержка схем будет описана далее в этой главе.

Последовательность представляет собой... да, последовательность объектов. Этими объектами могут быть узлы документа XML или простые значения вида `'June'` или `3.14`. Последовательность обладает определенным порядком и длиной. Вы можете использовать функции XPath 2.0 для определения количества объектов в последовательности, выборки подмножества объектов последовательности и вставки/удаления объектов в конкретной позиции последовательности. Пример переменной, содержащей последовательность:

```
<!-- sequences1.xsl -->
...
<xsl:variable name="months" as="xs:string*"
  select="('January', 'February', 'March', 'April',
           'May', 'June', 'July', 'August',
           'September', 'October', 'November',
           'December')"/>
```

Здесь стоит обратить внимание на ряд обстоятельств. Прежде всего заметьте, что мы используем новый атрибут `as` для определения типа данных переменной. Звездочка означает, что переменная может иметь произвольное количество значений. Если бы переменная была определена с типом данных `xs:string`, то переменная `$months` могла бы содержать только одно значение `xs:string`. Попытка выполнения этого фрагмента кода привела бы к фатальной ошибке.

Все значения в последовательности являются атомарными. Атомарным называется простое значение (в отличие от узла). Последовательность может содержать атомарные значения, как в приведенном при-

мере; также в ней может использоваться XPath для выбора узлов из документа XML. В следующем примере создается последовательность с атомарными значениями и информацией из документа XML:

```
<!-- sequences2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="cities" as="xs:string*">
      <xsl:sequence select="addressbook/address/city"/>
      <xsl:sequence select="('London', 'Adelaide', 'Rome')"/>
      <xsl:sequence select="('Jakarta', 'Sao Paulo', 'Timbuktu')"/>
    </xsl:variable>
    <xsl:text>Our customers live in these cities:&#xA;&#xA;</xsl:text>
    <xsl:value-of select="$cities" separator="&#xA;"/>
  </xsl:template>

</xsl:stylesheet>
```

Обратите внимание: переменная содержит элемент <xsl:sequence> для выбора элементов <city>, за которым следуют два элемента <xsl:sequence> для выбора трех строк в каждом. При использовании со следующим документом значение последовательности $cities содержит 12 объектов:

```
<!-- names.xml -->
<addressbook>

  <address>
    <name>
      <title>Mr.</title>
      <first-name>Chester Hasbrouck</first-name>
      <last-name>Frisby</last-name>
    </name>
    <street>1234 Main Street</street>
    <city>Sheboygan</city>
    <state>WI</state>
    <zip>48392</zip>
  </address>
...
    <city>Skunk Haven</city>
...
    <city>Winter Harbor</city>
...
    <city>Skunk Haven</city>
...
    <city>Boylston</city>
...
    <city>Lynn</city>
```

```
    <state>MA</state>
    <zip>02930</zip>
  </address>
</addressbook>
```

Выходные данные таблицы стилей выглядят так:

```
Our customers live in these cities:

Sheboygan
Skunk Haven
Winter Harbor
Skunk Haven
Boylston
Lynn
London
Adelaide
Rome
Jakarta
Sao Paulo
Timbuktu
```

Последовательность содержит 12 объектов: 6 объектов из элементов `<city>` документа XML и 6 значений из элементов `<xsl:sequence>`, содержащих строки. Последовательность объявляется с типом данных `xs:string*`, который означает, что *все объекты последовательности преобразуются в значения* `xs:string`. Хотя для выбора узлов используется шаблон `/addressbook/address/city`, содержимое последовательности представляет собой строки. Если перейти в последовательности к типу данных `item()*`, таблица стилей будет работать иначе:

```
<!-- sequences3.xsl -->
...
    <xsl:variable name="cities" as="item()*">
      <xsl:sequence select="addressbook/address/city"/>
      <xsl:sequence select="('London', 'Adelaide', 'Rome')"/>
      <xsl:sequence select="('Jakarta', 'Sao Paulo', 'Timbuktu')"/>
    </xsl:variable>
    <xsl:for-each select="$cities">
      <xsl:choose>
        <xsl:when test=". instance of element()">
          <xsl:text>        Node: </xsl:text>
        </xsl:when>
        <xsl:otherwise>
          <xsl:text>Atomic value: </xsl:text>
        </xsl:otherwise>
      </xsl:choose>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
...
```

Теперь последовательность содержит 12 объектов, 6 из которых являются узлами элементов, а еще 6 – строками. Вместо того чтобы просто записать значение каждого объекта в последовательность, мы воспользуемся новым оператором `instance of`, чтобы проверить, является ли объект элементом или атомарным значением. Результат выглядит так:

```
Our customers live in these cities:

        Node: Sheboygan
        Node: Skunk Haven
        Node: Winter Harbor
        Node: Skunk Haven
        Node: Boylston
        Node: Lynn
Atomic value: London
Atomic value: Adelaide
Atomic value: Rome
Atomic value: Jakarta
Atomic value: Sao Paulo
Atomic value: Timbuktu
```

Пути доступа

Одним из самых распространенных применений XPath является создание *путей доступа*. Путь доступа (location path) описывает местонахождение чего-либо в документе XML. Например, шаблон */addressbook/address/city* описывает местонахождение выбираемых элементов. Сначала мы используем пути доступа для поиска частей документов XML, а затем выполняем операции с ними при помощи выражений XPath. Но прежде чем знакомиться с тонкостями путей доступа, необходимо обсудить важное понятие *контекста*.

Контекст

Контекст относится к числу важнейших концепций XPath. Все, что делается в XPath, интерпретируется применительно к контексту. Документ XML можно рассматривать по аналогии с иерархией каталогов в файловой системе. Скажем, в приведенном ранее примере можно представить, что `sonnet` является каталогом корневого уровня файловой системы. В свою очередь, каталог `sonnet` содержит каталоги с именами `auth:author`, `title` и `lines`. В этой аналогии контекст соответствует текущему каталогу. Если я открою приглашение командной строки и выполню команду (скажем, `dir *.xsl`), то получу разные результаты в зависимости от текущего каталога. Аналогичным образом результаты вычисления выражений XPath обычно изменяются в зависимости от контекста.

[1.0] Контекст XPath 1.0

Чаще всего контекст можно рассматривать как узел дерева, который является отправной точкой для вычисления выражения. Если выражаться точнее, контекст образуется из пяти компонентов:

- *Контекстный узел* («текущий каталог»). Выражение XPath вычисляется, начиная с этого узла.
- Два целых числа: *позиция контекста* и *размер контекста*. Эти числа играют важную роль при обработке группы узлов. Например, можно написать выражение XPath для выбора всех элементов `<li>` заданного документа. Размер контекста относится к количеству элементов `<li>`, выбираемых данным выражением, а позиция контекста – к позиции элемента `<li>`, обрабатываемого в данный момент.
- Набор *переменных*, который включает имена и значения всех переменных, находящихся в области видимости.
- Набор всех *функций*, доступных для выражений XPath. Одни из этих функций определяются непосредственно в стандартах XPath и XSLT; другие (функции расширения) определяются программистом, создавшим таблицу стилей. (Функции расширения более подробно рассматриваются в главе 9.)
- Набор *объявлений пространств имен* в области видимости.

Впрочем, в большинстве случаев нас интересует только контекстный узел, а обо всем остальном можно забыть. Воспользуемся аналогией с командной строкой еще раз: когда вы находитесь в командной строке, у вас имеется текущий каталог; кроме того, (в зависимости от операционной системы) в системе может быть определено несколько переменных среды. Но при выполнении большинства команд важен только текущий каталог, а на переменные среды можно не обращать внимания.

[2.0] Контекст XPath 2.0

Как и следовало ожидать, концепция контекста в XPath 2.0 усложнилась. С введением поддержки XML Schema, даты, времени, упорядочения и функций XPath 2.0 приходится отслеживать намного больший объем информации.

В спецификации говорится о двух разных понятиях: *статическом контексте* и *динамическом контексте*. Статический контекст образуется из информации, доступной до вычисления выражения. Эта информация не изменяется в ходе вычисления выражения. Базовый URI-адрес, упорядочение по умолчанию, переменные, пространства имен и схемы в области видимости – все это входит в статический контекст. Контекстный объект, позиция контекста и размер контекста являются составляющими динамического контекста. В большинстве случаев нас интересует динамический контекст, но при необходимости можно обратиться к любой части контекста.

Статический контекст содержит следующую информацию:

- Работает ли процессор в *режиме совместимости с XPath 1.0*. Значение равно `true` или `false`.
- Набор *статически известных пространств имен*. Каждый элемент набора состоит из префикса и URI-адреса пространств имен.
- *Пространство имен по умолчанию для элементов и типов* – либо URI-адрес пространства имен, либо «none».
- *Пространство имен по умолчанию для функций* – либо URI-адрес пространства имен, либо «none».
- Набор *схемных определений в области видимости* – типы, элементы и атрибуты схемы, находящиеся в области видимости.
- Набор *переменных* в области видимости.
- *Статический тип контекстного объекта*.
- *Сигнатуры функций в области видимости*, включая пространство имен и «арность» (количество параметров) каждой функции, а также статические типы параметров и результатов. В набор входят функции-конструкторы.
- *Статически известные упорядочения*. Формат упорядочений определяется реализацией.
- *Упорядочение по умолчанию*.
- *Базовый URI-адрес*.
- Набор *статически известных документов*. Речь идет о документах, доступных в результате вызова `doc()`. За дополнительной информацией о функции `doc()` обращайтесь к разделу «[2.0] Функции doc() и doc-available()» главы 8.
- Набор *статически известных коллекций*. За дополнительной информацией о коллекциях обращайтесь к описанию функции [2.0] `collection()` в приложении C.
- *Статически известный тип коллекции по умолчанию*. По умолчанию используется тип коллекции `node()*`, если только процессору XSLT не было задано другое значение. За дополнительной информацией о коллекциях также обращайтесь к описанию функции [2.0] `collection()` в приложении C.

Динамический контекст содержит всю информацию статического контекста, а также:

- *Контекстный объект* – аналог контекстного узла XPath 1.0 с тем различием, что контекстный объект может быть атомарным значением. В XPath 1.0 всегда можно запросить родителя контекстного узла. Если вы попытаетесь сделать это в XPath 2.0 и контекстный объект не является узлом, процессор выдаст исключение.
- *Позиция контекста* – работает так же, как в XPath 1.0; функция `position()` возвращает позицию контекста.

- *Размер контекста* – работает так же, как и в XPath 1.0; функция `last()` возвращает размер контекста.
- Набор *значений переменных* в области видимости.
- Набор *реализаций функций* в области видимости.
- *Текущая дата/время*. Значение не изменяется во время выполнения выражения.
- *Неявно заданный часовой пояс*. Для получения значения используется новая [2.0] функция `implicit-timezone()` (см. приложение C).
- Набор *доступных документов*, для обращения к которым используется функция [2.0] `doc()` (см. приложение C).
- Набор *доступных коллекций*, для обращения к которым используется функция [2.0] `collection()` (см. приложение C).
- *Коллекция по умолчанию*, возвращаемая функцией [2.0] `collection()` без аргументов.

Простые пути доступа

Итак, теперь вы знаете, что такое контекст и почему он важен, и мы можем перейти к практическому рассмотрению путей доступа. Начав с пары простых примеров, будем постепенно продвигаться к более сложным путям доступа, использующим нетривиальные возможности XPath. Ранее уже приводился один из простейших шаблонов XSLT:

```
<xsl:template match="/">
```

Шаблон выбирает корневой узел документа. Другое простое выражение XPath встречалось нам в элементе `<xsl:value-of>`:

```
<xsl:value-of select="."/>
```

Шаблон выбирает контекстный узел, представленный точкой. В завершение обзора простейших путей доступа следует упомянуть двойную точку (`..`), обозначающую *родителя* контекстного узла:

```
<xsl:value-of select=".."/>
```

Все эти выражения XPath имеют одну общую особенность: в них не используются имена элементов. Как вы, возможно, заметили из нашего примера Hello World, имена элементов могут использоваться для выбора элементов с конкретным именем:

```
<xsl:apply-templates select="greeting"/>
```

В этом примере выбираются все элементы `<greeting>` в текущем контексте, и к каждому из них применяется подходящий шаблон. Возвращаясь к примеру `sonnet`, мы можем создать пути доступа, содержащие более одного уровня в иерархии документа:

```
<xsl:apply-templates select="lines/line"/>
```

В этом примере выбираются все элементы <line>, содержащиеся в элементах <lines> в текущем контексте. Если текущий контекст не содержит элементов <lines>, это выражение возвращает пустой набор узлов. Если текущий контекст содержит множество элементов <lines>, но ни один из них не содержит элементов <line>, выражение все равно вернет пустой набор узлов.

Относительные и абсолютные выражения

В спецификации XPath определяются два вида выражений XPath: *относительные* и *абсолютные*. В предыдущем примере используется относительное выражение XPath, так как определяемые в нем узлы зависят от текущего контекста. Абсолютное выражение XPath начинается с косой черты (/), которая приказывает процессору XSLT начинать с корня документа независимо от текущего контекста. Другими словами, абсолютные выражения XPath могут вычисляться от любого контекстного узла по вашему усмотрению; результат будет одним и тем же. Пример абсолютного выражения XPath:

```
<xsl:apply-templates select="/sonnet/lines/line"/>
```

Абсолютные выражения удобны тем, что вам не нужно беспокоиться о контекстных узлах. Другое преимущество – простота поиска процессором XSLT всех узлов, удовлетворяющих абсолютному выражению: в этом выражении фактически говорится, что в корне документа должен присутствовать элемент <sonnet>, содержащий, как минимум, один элемент <lines>, и по крайней мере один из этих элементов <lines> должен содержать хотя бы один элемент <line>. Если хотя бы одно из этих условий не выполняется, процессор XSLT может прервать поиск по дереву и вернуть пустой набор узлов.

Недостаток абсолютных выражений XPath заключается в том, что они усложняют повторное использование шаблонов. Оба следующих шаблона обрабатывают элементы <line>, но повторно использовать второй шаблон будет сложнее:

```
<xsl:template match="line">
  ...
</xsl:template>

<xsl:template match="/sonnet/lines/line">
  ...
</xsl:template>
```

Если второй шаблон содержит гениальный код обработки элементов <line>, а ваш документ содержит элементы <line>, не соответствующие абсолютному шаблону XSLT, использовать шаблон в готовом виде не удастся. (С другой стороны, процессору XSLT с шаблоном /sonnet/lines/line приходится выполнять меньший объем работы; он точно знает, где следует искать интересующие нас элементы.) Вообще абсолютные выражения следует использовать, только если вам необходи-

ма специальная обработка для конкретного случая. Если элементы <line>, соответствующие шаблону /sonnet/lines/line, должны обрабатываться особым образом, используйте абсолютный шаблон в атрибуте match. Если особая обработка таких элементов <line> не нужна, не используйте абсолютный шаблон. Учитывайте это обстоятельство при проектировании шаблонов.

Выбор других объектов в путях доступа

До настоящего момента мы рассматривали выражения XPath, использовавшие либо имена элементов (/sonnet/lines/line), либо специальные символы (/ или ..) для выбора элементов в документе XML. Разумеется, документы XML могут содержать и другие объекты, помимо элементов; в этом разделе речь пойдет о том, как выбирать такие объекты.

Выбор атрибутов

Чтобы выбрать атрибут, поставьте знак @ перед именем атрибута. В нашем примере атрибут type элемента <sonnet> выбирается выражением XPath /sonnet/@type. Если контекстным узлом является сам элемент <sonnet>, относительное выражение XPath @type делает то же самое.

Выбор текста элемента

Чтобы выбрать текст элемента, просто включите ссылку на него в свое выражение. Элемент <xsl:value-of select="/sonnet/auth:author/last-name"/> в нашем примере возвращает Shakespeare. Также можно воспользоваться функцией string(), хотя обычно это необязательно.

Следующие инструкции XSLT:

```
<xsl:value-of select="/sonnet/something_else:author/first-name"/>
<xsl:text> </xsl:text>
<xsl:value-of select="/sonnet/something_else:author/last-name/string()"/>
```

генерируют следующий результат:

```
William Shakespeare
```

Учтите, что получение текста элемента, имеющего дочерние элементы, может привести к неожиданному результату. Например, элемент <xsl:value-of select="/sonnet/auth:author"/> вернет строку Shakespeare-WilliamBritish15641616. Тексты всех потомков узла <auth:author> объединяются посредством конкатенации. Чтобы придать этим узлам более привлекательный вид, их придется обрабатывать по отдельности.

Наконец, существует тест узлов text(), который выбирает все текстовые узлы, являющиеся потомками контекстного объекта. Впрочем, вам практически никогда не потребуется применять его на практике – получение текста способом, описанным выше, гораздо удобнее.

Выбор узлов комментариев, инструкций по обработке и пространств имен

К этому моменту мы рассмотрели большую часть составляющих документов, которые вам когда-либо придется выбирать в выражениях XPath. Существует пара других тестов узлов XPath для описания частей документов XML. Тесты `comment()` и `processing-instruction()` позволяют выбирать комментарии и инструкции по обработке в документах XML. Вернемся к нашему примеру: выражение XPath `/processing-instruction()` возвращает две инструкции (с именами `xml-stylesheet` и `cocoon-process`). Выражение `/sonnet/comment()` возвращает узел комментария, начинающийся со слов «Is there an official title for this sonnet?» (Это единственный комментарий в самом элементе `<sonnet>`; остальные комментарии находятся за пределами корневого элемента.)

Подобный способ обработки узлов комментариев весьма полезен. Если в документе XML присутствуют комментарии, тест узлов `comment()` позволит выводить комментарии только там, где потребуется. Для этого можно использовать следующий шаблон XSLT:

```
<xsl:template match="comment()">
  <span class="comment">
    <p><xsl:value-of select="."/></p>
  </span>
</xsl:template>
```

Например, в таблице стилей можно определить атрибуты CSS для вывода комментариев крупным жирным фиолетовым шрифтом. Чтобы удалить все комментарии из выходного документа, откройте таблицу стилей и измените шаблон, обрабатывающий комментарии, так, чтобы он ничего не делал. (Также можно просто удалить его; шаблон для комментариев по умолчанию не делает ничего.)

В XPath существует еще одна разновидность узлов – редко используемые *узлы пространств имен*. Для получения узлов пространств имен необходимо использовать *ось пространств имен*; эта тема описана в разделе «Оси» далее в этой главе. Одно замечание по поводу узлов пространств имен, если вам вдруг когда-либо придется их использовать: при идентификации узлов пространств имен префикс пространства не учитывается. В нашем примере `sonnet` используется префикс пространства имен `auth`, отображаемый на значение `http://www.authors.com/`. Если таблица стилей использует префикс пространства имен `writers` для ссылок на тот же URL-адрес, выражение XPath `/sonnet/writers:*` вернет элемент `<auth:authors>`. Несмотря на различающиеся префиксы пространств имен, URL-адреса, на которые они ссылаются, совпадают. Скорее всего, префикс пространства имен будет представлять интерес в единственной ситуации: при поиске конкретного узла пространства имен. Имя узла пространства имен является префиксом. Почти во всех остальных случаях для поиска нужной информации используется URI-адрес пространства имен.

Впрочем, несмотря на все сказанное, шансы на то, что вам когда-либо потребуется работать с узлами пространств имен, невелики.

Метасимволы

В XPath используются три метасимвола:

Звездочка (*)

Выбирает все узлы элементов в текущем контексте. Учтите, что метасимвол `*` выбирает только узлы элементов; узлы атрибутов, текста, комментариев или инструкций по обработке не включаются. Со звездочкой также может использоваться префикс пространства имен. В нашем примере `sonnet` выражение XPath `auth:*` возвращает все узлы элементов из текущего контекста, связанные с пространством имен `http://www.authors.com/`.

[2.0] XPath 2.0 позволяет использовать метасимвол в качестве префикса пространства имен. Выражение XPath `*:authors` возвращает все узлы элементов из текущего контекста с именем `author` независимо от их пространства имен. В XPath 2.0 оба варианта записи `auth:*` и `*:author` допустимы; в XPath 1.0 поиск по шаблону `*:author` приводит к фатальной ошибке.

Как обычно, поиск соответствий в пространствах имен осуществляется по URL-адресу пространства, а не по префиксу. Допустим, пространство имен `http://www.authors.com/` связывается в документе XML с префиксом `auth`, а в таблице стилей – с префиксом `something_else`. Поиск по шаблону `something_else:*` в таблице стилей вернет все документы с префиксом `auth` из документа XML.

Знак «коммерческое at» со звездочкой (@)*

Выбирает все узлы атрибутов в текущем контексте. Префикс пространства имен может использоваться с метасимволом атрибута. В нашем примере `sonnet` запись `@auth:*` возвращает все узлы атрибутов в текущем контексте, связанные с URL-адресом пространства имен `http://www.authors.com`. (С пространством имен `auth` не связаны никакие атрибуты; если бы мы добавили элемент вида `<first-name auth: nickname="Bill">William</first-name>`, то шаблон `auth:nickname` соответствовал бы этому выражению.)

[2.0] Как и в случае с элементами, XPath 2.0 позволяет использовать метасимвол в качестве префикса пространства имен. В XPath 2.0 оба варианта записи `@auth:*` и `@*:nickname` допустимы; в XPath 1.0 использование метасимвола в качестве префикса пространства имен приводит к фатальной ошибке.

Тест узлов node( )

Выбирает все узлы в текущем контексте независимо от их типа. В частности выбираются узлы элементов, текста, комментариев, инструкций по обработке, атрибутов и пространств имен.

Кроме перечисленных метасимволов в XPath поддерживается двойная косая черта (//), которая обозначает нуль и более элементов. Например, выражение XPath `//line` выбирает все элементы `<line>` независимо от их местоположения в документе. Это абсолютное выражение XPath, потому что оно начинается с косой черты. Двойная косая черта также может использоваться в любой точке выражения XPath; так, выражение `/sonnet/descendant-or-self::node()/line` выбирает все элементы `<line>`, которые являются потомками элемента `<sonnet>` в корне документа XML. Выражения `/sonnet//line` и `/sonnet/descendant-or-self::node()/line` эквивалентны. (Запись `descendant-or-self` является *осью*; оси рассматриваются в следующем разделе.)

Двойная косая черта (//) – чрезвычайно мощный оператор, однако следует помнить, что он серьезно снижает эффективность ваших таблиц стилей. Встретив выражение XPath `//line`, процессор XSLT должен обратиться к каждому узлу документа и проверить, имеются ли в нем элементы `<line>`. Чем конкретнее записаны выражения XPath, тем меньше работы придется выполнять процессору XSLT, и тем быстрее будут обрабатываться таблицы стилей. Вернемся к метафоре с файловой системой: если я открою командную строку Windows и введу команду `dir/s c:\*.xsl`, операционной системе придется просмотреть все подкаталоги, чтобы найти все файлы **.xsl*. Но если ввести команду `dir/s c:\doug\projects\stylesheets\*.xsl`, операционной системе необходимо будет просмотреть меньшее количество подкаталогов, и команда будет выполнена намного быстрее.

Оси

Итак, теперь вы знаете, как выбирать дочерние элементы, атрибуты, текст, комментарии и инструкции по обработке при помощи несложных выражений XPath. Конечно, на практике может возникнуть необходимость в выборе других объектов, например:

- Всех предков контекстного узла
- Всех потомков контекстного узла
- Всех предшествующих или следующих узлов того же уровня (то есть узлов с тем же родителем)

Для реализации выбора различных коллекций узлов в XPath используются *оси* (*axes*). Всего существует 13 разных осей; мы обсудим все оси, хотя большинство из них вам вряд ли понадобится. Чтобы использовать ось в выражении XPath, укажите имя оси, двойное двоеточие (`::`) и имя выбираемого элемента (если оно есть).

Но прежде чем приводить определения всех осей, необходимо сначала разобраться с полным синтаксисом XPath.

Полный синтаксис

Во всех выражениях XPath, рассматривавшихся до настоящего момента, использовался *сокращенный синтаксис* XPath. Чаще всего используется именно эта разновидность синтаксиса; тем не менее большинство редко используемых осей задается только в полном варианте синтаксиса. Например, в записи выражения XPath для выбора всех элементов `<line>` в текущем контексте мы использовали сокращенный синтаксис:

```
<xsl:apply-templates select="line"/>
```

Если вам не лень вводить лишние символы, используйте полный синтаксис и укажите, что вам нужны все потомки `<line>` текущего контекста:

```
<xsl:apply-templates select="child::line"/>
```

А сейчас я перечислю все оси и укажу, какие из них существуют в сокращенном синтаксисе.

Перечень осей

Далее перечислены все оси, определенные в стандарте XPath, с кратким описанием каждой из них:

`child`

Дочерние узлы контекстного узла. Как упоминалось ранее, выражение XPath `child::lines/child::line` эквивалентно `lines/line`. Если выражение XPath (например, `sonnet`) не содержит спецификатора оси, по умолчанию используется ось `child`. К дочерним узлам контекстного узла относятся все узлы комментариев, элементов, инструкций по обработке и текста. Узлы атрибутов и пространств имен не считаются дочерними узлами контекстного узла.

`parent`

Родитель контекстного узла, если он существует. (Если контекстный узел является корневым, ось `parent` возвращает пустой набор узлов.) В выражениях XPath родительская ось может быть представлена сокращением `..` (две точки), обозначающим переход к родителю текущего узла. Если элементы `<first-name>` и `<last-name>` являются дочерними по отношению к элементу `<author>`, а элемент `<first-name>` является контекстным узлом, выражения `../last-name`, `parent::author/last-name` и `parent::*/last-name` эквивалентны. Если контекстный узел не имеет родителя, ось возвращает пустой набор узлов.

`self`

Содержит сам контекстный узел. В выражениях XPath ось `self` может быть представлена сокращением `.` (точка). Выражения `.`, `self::node()` и `self::*` в XSLT 1.0 эквивалентны.

[2.0] В XSLT 2.0 ось `self` выбирает контекстный *объект*, который может и не быть узлом. Если контекстный объект является атомарным значением, выражения `self::node()` и `self::*` приводят к выдаче ошибки процессором XSLT. В этом случае к оси `self` можно обратиться только с использованием точки. Если контекстный объект является узлом, ось `self` работает так же, как в XSLT 1.0.

`attribute`

Содержит атрибуты контекстного узла. Если контекстный узел не является узлом элемента, ось пуста. Ось `attribute` может быть представлена сокращенным обозначением `@`. Выражения `attribute::type` и `@type` эквивалентны.

`ancestor`

Содержит родителя контекстного узла, родителя родителя и т. д. Ось `ancestor` всегда содержит корневой узел, если только контекстный узел не является корневым.

`ancestor-or-self`

Содержит контекстный узел, его родителя, родителя родителя и т. д. Ось всегда содержит корневой узел.

`descendant`

Содержит всех потомков контекстного узла, всех потомков потомков контекстного узла и т. д. К потомкам относятся все узлы комментариев, элементов, инструкций по обработке и текста, находящиеся под контекстным узлом. Другими словами, ось `descendant` не включает узлы атрибутов и пространств имен. (Как упоминалось ранее, хотя узлы атрибутов имеют родительский узел элемента, узел атрибута не считается его дочерним узлом.)

`descendant-or-self`

Содержит контекстный узел, его потомков, потомков потомков и т. д. Как обычно, к потомкам относятся все узлы комментариев, элементов, инструкций по обработке и текста; узлы атрибутов и пространств имен не включаются.

`preceding-sibling`

Содержит все предшествующие узлы одного уровня с контекстным узлом – иначе говоря, все узлы, имеющие общего родителя с контекстным узлом, но находящиеся перед ним в документе XML. Если контекстный узел является узлом атрибута или пространства имен, ось `preceding-sibling` пуста.

`following-sibling`

Содержит все последующие узлы одного уровня с контекстным узлом – иначе говоря, все узлы, имеющие общего родителя с контекстным узлом, но следующие после него в документе XML. Если контекстный узел является узлом атрибута или пространства имен, ось `following-sibling` пуста.

`preceding`

Содержит все узлы, предшествующие контекстному узлу в документе, кроме предков, узлов атрибутов и пространств имен.

`following`

Содержит все узлы, следующие за контекстным узлом в документе, кроме потомков, узлов атрибутов и пространств имен.

`namespace`

Содержит узлы пространств имен контекстного узла. Если контекстный узел не является узлом элемента, эта ось пуста.

Предикаты

У выражений XPath имеется еще один аспект, который нами еще не обсуждался: *предикаты*. Они представляют собой фильтры, ограничивающие состав узлов, выбираемых выражениями XPath. Каждый предикат вычисляется и преобразуется в логическое значение (`true` или `false`). Если для узла предикат равен `true`, узел выбирается, а если предикат равен `false` – игнорируется. Предикаты всегда записываются в квадратных скобках (`[]`). Пример:

```
<xsl:apply-templates select="line[position() = 7]"/>
```

Выражение выбирает седьмой элемент `<line>` в текущем контексте. Если текущий контекст содержит шесть или менее элементов `<line>`, выражение XPath возвращает пустой набор узлов. Предикаты могут строиться из нескольких разных компонентов; все они описаны ниже.

Числа в предикатах

Вместо функции `position()` можно использовать число. Например, выражение XPath `line[7]` выбирает седьмой элемент `<line>` в контекстном узле; оно в точности эквивалентно `line[position()=7]`. XPath поддерживает логические операторы `and` и `or`, а также оператор объединения предикатов (`|`). Выражение `line[position()=3 and @style]` соответствует всем элементам `<line>`, находящимся в третьей позиции и обладающим атрибутом `style`, тогда как `line[position()=3 or @style]` соответствует всем элементам `<line>`, которые находятся в третьей позиции *или* обладают атрибутом `style`.

При желании можно использовать сразу несколько предикатов; `line[3][@style]` или `line[@style][3]` – оба варианта записи допустимы. Тем не менее эквивалентными они не являются. Предикаты обрабатываются слева направо. Процессор XSLT обрабатывает первый шаблон, выбирая все узлы `<line>`, которые находятся в третьей позиции набора одноуровневых узлов `<line>`, а затем выбирает из них все узлы с атрибутом `style`. Во втором шаблоне процессор выбирает все элементы `<line>` с атрибутом `style`, а затем выбирает третий узел в этой последовательности. Первый шаблон может соответствовать произвольному

количеству узлов, а второй никогда не соответствует более чем одному. В общем случае первый предикат фильтрует узлы, второй фильтрует набор, прошедший первый предикат, третий предикат фильтрует узлы, прошедшие второй предикат, и т. д.

Функции в предикатах

Помимо чисел в предикатах могут использоваться функции XPath и XSLT. Приведем несколько примеров:

`line[last()]`

Выбирает последний элемент `<line>` в текущем контексте.

`line[position() mod 2 = 0]`

Выбирает все элементы `<line>` с четными номерами. (Оператор `mod` возвращает остаток от деления; при делении позиций всех четных элементов на 2 остаток равен 0.)

`sonnet[@type="Shakespearean"]`

Выбирает все элементы `<sonnet>`, у которых атрибут `type` имеет значение `Shakespearean`. Использовать кавычки вместо апострофов не обязательно; это выражение XPath совпадет как с `<sonnet type="Shakespearean">`, так и с `<sonnet type='Shakespearean'>`.

`ancestor::table[@border="1"]`

Выбирает всех предков узла `<table>` текущего контекста с атрибутом `border`, равным 1.

`count(/body/table[@border="1"])`

Возвращает количество элементов `<table>` с атрибутом `border`, равным 1, которые являются дочерними по отношению к элементам `<body>`, являющимся дочерними узлами коренного узла. Обратите внимание на использование предиката в составе пути доступа.

Шаблоны значений атрибутов

Хотя формально шаблоны значений атрибутов определяются в спецификации XSLT (XSLT 1.0 – раздел 7.6.2, XSLT 2.0 – раздел 5.6), мы обсудим их здесь. Шаблон значения атрибута (иногда используется сокращение AVT, от Attribute Value Template) представляет собой выражение XPath, результат вычисления которого замещает исходное выражение. Например, можно создать элемент таблицы HTML `<table>` следующего вида:

```
<table border="{@size}"/>
```

В этом примере вычисляется выражение XPath `@size`, и его значение, каким бы оно ни было, вставляется в выходное дерево как значение атрибута `border`. Шаблоны значений атрибутов могут использоваться в любых элементах таблицы стилей с литеральным результатом (на-

пример, в элементах HTML и других конструкциях, не принадлежащих пространству имен XSLT). Шаблоны значений атрибутов также могут использоваться в следующих атрибутах XSLT:

- Атрибуты `name` и `namespace` элемента `<xsl:attribute>`
- Атрибуты `name` и `namespace` элемента `<xsl:element>`
- Атрибуты `format`, `lang`, `letter-value`, `grouping-separator` и `grouping-size` элемента `<xsl:number>`
- Атрибут `name` элемента `<xsl:processing-instruction>`
- Атрибуты `lang`, `data-type`, `order` и `case-order` элемента `<xsl:sort>`

[2.0] XSLT 2.0 разрешает использовать AVT в ряде дополнительных мест:

- Атрибуты `regex` и `flags` нового элемента *[2.0]* `<xsl:analyze-string>`
- Атрибуты `name`, `namespace` и `separator` элемента `<xsl:attribute>` (как в XSLT 1.0)
- Атрибуты `name` и `namespace` элемента `<xsl:element>` (как в XSLT 1.0)
- Атрибут `collation` нового элемента *[2.0]* `<xsl:for-each-group>`
- Атрибут `terminate` элемента `<xsl:message>`
- Атрибут `name` элемента *[2.0]* `<xsl:namespace>`
- Атрибуты `format`, `lang`, `letter-value`, `ordinal`, `grouping-separator` и `grouping-size` элемента `<xsl:number>` (`ordinal` – новый атрибут XSLT 2.0; остальные атрибуты остались неизменными с XSLT 1.0)
- Атрибут `name` элемента `<xsl:processing-instruction>`
- Атрибуты `format`, `href`, `method`, `byte-order-mark`, `cdata-section-elements`, `doctype-public`, `doctype-system`, `encoding`, `escape-uri-attributes`, `include-content-type`, `indent`, `media-type`,`normalization-form`, `omit-xml-declaration`, `standalone`, `undeclare-prefixes` и `output-version` нового элемента *[2.0]* `<xsl:result-document>`
- Атрибуты `lang`, `order`, `collation`, `stable`, `case-order` и `data-type` элемента `<xsl:sort>` (`collation` и `stable` – новые атрибуты XSLT 2.0)
- Атрибут `separator` элемента `<xsl:value-of>` (новый атрибут XSLT 2.0)

Типы данных

Одним из важнейших дополнений XPath 2.0 стала поддержка системы типов данных XML Schema. Модели данных XPath 1.0 и 2.0 так сильно различаются, что мы рассмотрим их в разных разделах. В общем случае большинство команд, работавших в XPath 1.0, работает и в XSLT 2.0. С другой стороны, все команды XPath 2.0, использующие новые возможности типизации данных, в XPath 1.0 работать не будут.

Типы данных XPath 1.0

В XPath 1.0 выражение возвращает один из четырех типов данных:

`node-set`

Представляет набор узлов (пустой или содержащий произвольное количество узлов).

`boolean`

Представляет значение `true` или `false`. Учтите, что строки `true` и `false` не имеют специальной интерпретации или значения в XPath; за более подробной информацией обращайтесь к разделу «Преобразование к логическим значениям» главы 5.

`number`

Представляет вещественное число. Все числа в XPath и XSLT реализуются как вещественные числа с плавающей точкой; тип данных `integer` (или `int`) в XPath и XSLT не существует. Говоря конкретнее, все числа реализуются в стандарте вещественных чисел IEEE 754; этот же стандарт используется для примитивных типов Java `float` и `double`. Кроме обычных чисел, существуют пять специальных числовых значений: положительная и отрицательная бесконечность, положительный и отрицательный нуль и `NaN` («нечисло» – специальное обозначение для всего, что не является числом).

`string`

Представляет нуль или более символов в соответствии с определением из спецификации XML.

Эти типы данных обычно просты, и преобразования между ними (кроме наборов узлов) достаточно прямолинейны. Здесь эти типы данных подробно не обсуждаются; мы рассмотрим их там, где это потребуется для решения конкретных задач.

Типы данных XPath 2.0

Пожалуй, самые значительные изменения при написании таблиц стилей XSLT 2.0 связаны с моделью данных XPath 2.0. Мы рассмотрим типы данных, поддерживаемые XPath 2.0. XPath 2.0 поддерживает все базовые типы данных, определяемые в XML Schema, а схемосовместимый процессор XSLT 2.0 позволит вам создавать собственные типы данных. Начнем с основных типов данных: *базовый процессор XSLT поддерживает только эти типы*. Для поддержки других типов, включая типы данных, определяемые пользователем (например, `po:purchaseOrder`), и производные типы, определяемые в XML Schema (например, `xs:nonNegativeInteger`), потребуется схемосовместимый процессор XSLT.

Вы уже видели, как элемент `<xsl:variable name="sample" select="'3'" as="xs:integer"/>` используется для создания значения `xs:integer`.

XPath 2.0 также поддерживает *функции-конструкторы*, перечисленные в следующем ниже списке. Например, элемент `<xsl:variable name="sample" select="xs:integer(3)"/>` создает новое значение `xs:integer`, а элемент `<xsl:variable name="birthday" select="xs:date('1995-04-21')"/>` создает новое значение `xs:date`:

`xs:string`

Тип данных `xs:string` представляет строку. Все типы данных, поддерживаемые XPath 2.0, имеют *строковое представление*. Если вы хотите знать, как тип данных выглядит в строковом формате, элемент XSLT `<xsl:value-of>` поможет вам в этом. Любые данные можно преобразовать в строку функцией-конструктором `xs:string()`. Конструктор является аналогом метода Java `toString()`, наследуемого всеми классами.

`xs:boolean`

Как и в XSLT 1.0, строковые значения `true` и `false` не имеют специального смысла. Для работы с логическими данными XPath предоставляет функции `true()` и `false()`, возвращающие соответствующие значения. За дополнительной информацией об этих значениях обращайтесь к разделу «Преобразование к логическим значениям» главы 5. Для создания логических значений также можно воспользоваться конструктором `xs:boolean()`. Вызов `xs:boolean(1)` создает значение `true`, а `xs:boolean(0)` создает значение `false`.

`xs:decimal`

XML Schema определяет значение `xs:decimal` как числовое значение, состоящее из десятичных цифр (от `0` до `9`) и начинающееся с необязательного плюса или минуса. *Значение* `xs:decimal` *не может содержать экспоненциальную часть*. Спецификация XML Schema указывает, что реализация должна поддерживать, как минимум, 18 десятичных цифр. Значения 42, 8.37284 и -83982.22 – примеры допустимых значений типа `xs:decimal`.

`xs:float` *и* `xs:double`

Типы данных `xs:float` и `xs:double` базируются на вещественных типах IEEE с одинарной и двойной точностью соответственно. В отличие от `xs:decimal`, значения `xs:float` и `xs:double` могут содержать экспоненциальную часть. У типов `xs:float` и `xs:double` имеются три специальных значения: `INF` (бесконечность), `-INF` (отрицательная бесконечность) и `NaN` (нечисло). Значения 42, 8.37284, -83982.22, -8.39822e4, INF и -0 – примеры допустимых значений типов `xs:float` или `xs:double`.

`xs:integer`

Целое число, не содержащее десятичной точки или дробной части. Целое число может иметь признак положительного или отрицательного значения (+ или -); без знака `xs:integer` считается положительным числом.

`xs:duration`

Тип `xs:duration` представляет временной промежуток. Значение состоит из шести компонентов: год, месяц, день, час, минуты и секунды. В спецификации XML Schema указано, что реализация должна поддерживать, по меньшей мере, год из четырех цифр и значение секунд с точностью до трех цифр в дробной части (точность до миллисекунд). Промежуток в 1 день 7 месяцев 18 дней 4 часа 27 минут и 3.673 секунды записывается в виде `P1Y7M18DT4H27M3.673S`.

Значения даты и времени

Существуют три типа данных для представления даты и времени: `xs:date`, `xs:time` и `xs:dateTime`. Дата задается в формате `ГГГГ-ММ-ДД`; так, запись `1995-04-21` соответствует 21 апреля 1995 года. Время задается в формате `чч:мм:сс.ссс`; следовательно, запись `19:38:55.183` соответствует 55.183 секундам после 19:38.

Значения `xs:date` и `xs:time` содержат необязательный индикатор часового пояса. Часовой пояс со знаком `+` или `-` задает смещение по отношению к всемирному координированному времени UTC (Universal Time Coordinated), также известному как гринвичское среднее время (GMT, Greenwich Mean Time). Например, зимой (когда не действует летнее время) на восточном побережье США действует часовой пояс `-05:00`. Учтите, что при нормализации процессором XSLT значения даты или времени компоненты значения могут измениться. Время `19:30:55.183-05:00` эквивалентно `00:30:55.183Z` (значение даты или времени, завершающееся символом `Z`, нормализовано по UTC).

Наконец, значение `xs:dateTime` представляет собой комбинацию `xs:date` и `xs:time`. В письменном представлении `xs:dateTime` компоненты разделяются буквой `T`. Например, значение `1995-04-21T19:38:55.183-05:00`, полученное в результате объединения предыдущих примеров, соответствует 55.183 секундам после 19:38 21 апреля 1995 года с отставанием на 5 часов от UTC. Это значение эквивалентно `1995-04-22T00:38:55.183Z`.

Учтите, что `xs:date`, `xs:time` и `xs:dateTime` могут иметь отрицательные значения.

Компоненты значений даты и времени

XML Schema определяет типы данных `xs:gYearMonth`, `xs:gYear`, `xs:gMonthDay`, `gDay` и `gMonth`. Примеры этих значений (в указанном порядке): `1995-04` – апрель 1995 года; `1995` – 1995 год; `--04-21` – 21 апреля; `---21` – 21 день месяца; `--04` – апрель.

`xs:hexBinary` *и* `xs:base64Binary`

Тип данных `xs:hexBinary` представляет собой строку, состоящую из двоичных октетов. Иначе говоря, строка должна состоять только из пар шестнадцатеричных цифр ([0-9a-fA-F]). Тип данных `xs:base64Binary` использует кодировку Base64 для представления двоичных

данных. Он также представляет собой строку. Значение xs:base64-Binary состоит из 65 символов, определенных в RFC2045: [0-9a-zA-Z], знака «плюс» (+), косой черты (/) и знака равенства (=), а также некоторых символов-пропусков. (Документ RFC доступен по адресу *http://www.ietf.org/rfc/rfc2045.txt*.)

xs:anyURI

Значение xs:anyURI представляет собой строку, которая является действительным URI-адресом в соответствии с определением из RFC 2396 (и последующим обновлением RFC2732). Документы RFC доступны по адресам *http://www.ietf.org/rfc/rfc2396.txt* и *http://www.ietf.org/rfc/rfc2732.txt*.

xs:QName

xs:QName (полное имя, Qualified Name) представляет собой имя XML, уточненное префиксом пространства имен. Например, auth:author является полным именем. Значение xs:QName может использоваться в таблице стилей только в том случае, если префикс пространства имен находится в области видимости.

xs:anyType *и* xs:anySimpleType

Тип данных xs:anyType считается базовым типом данных (*ur-mun* в спецификации типов данных XML Schema), на основе которого создаются все остальные произвольные типы. Значение xs:anyType может содержать произвольные данные; какие-либо ограничения отсутствуют.

Тип данных xs:anySimpleType представляет собой ограниченную версию xs:anyType; значение xs:anySimpleType может быть любым допустимым значением любого примитивного типа данных, определенного в XML Schema. *Примитивным типом данных* называется тип данных, не определяемый на основе другого типа. Тип данных xs:float относится к примитивным, поэтому значение xs:float будет относиться к xs:anySimpleType. С другой стороны, тип xs:integer примитивным не является; он определяется на базе xs:float.

Следующие типы данных были добавлены в спецификацию XPath 2.0 and XQuery 1.0 Data Model. Они поддерживаются наряду со всеми типами данных, определяемыми в XML Schema:

xs:yearMonthDuration *и* xs:dayTimeDuration

Эти два типа были добавлены в пространство имен типов данных XML Schema в спецификации XPath 2.0 and XQuery 1.0 Data Model. Они соответствуют двум половинам значения xs:duration. xs:yearMonthDuration содержит некоторое количество годов и месяцев, а xs:dayTimeDuration – некоторое количество дней, часов, минут и секунд. Оба типа данных могут иметь отрицательные значения. Временной промежуток в 12 лет и 2 месяца записывается в виде P12Y2M. Обратите внимание: xs:yearMonthDuration не содержит компонента

дней. Промежуток в 4 дня 7 часов 47 минут и 32.883 секунды записывается в виде `P4DT7H47M32.883S`.

`xs:untyped` *и* `xs:untypedAtomic`

Эти типы данных определяются в спецификации XPath 2.0 and XQuery 1.0 Data Model. Узел, который еще не был проверен, имеет динамический тип `xs:untyped`, а атомарное значение, которое еще не было проверено, имеет динамический тип `xs:untypedAtomic`.

`xs:anyAtomicType`

В XPath 2.0 и XQuery 1.0 все простые типы происходят от базового типа `xs:anyAtomicType`. Например, `xs:integer`, `xs:boolean` и `xs:string` являются производными от `xs:anyAtomicType`.

Операторы XPath

XPath поддерживает обширный набор операторов, расширяющих возможности выражений. Мы рассмотрим все операторы.

[1.0] В первых двух разделах обсуждаются все операторы XPath 1.0 кроме оператора объединения | (вертикальная черта). В XPath 1.0 объединение выполняется только этим оператором; новое ключевое слово `union` поддерживается только в XPath 2.0. За более подробной информацией об операторе объединения (|) и ключевом слове `union` обращайтесь к разделу «[2.0] Операторы множеств – except, intersect и union» далее в этой главе.

[2.0] В XPath 2.0 некоторые операторы могут использоваться с датой, временем и временными промежутками (например, к `xs:dayTimeDuration` можно прибавить 12 часов). Операторы и возможности, специфические для XPath 2.0, будут рассмотрены после операторов, общих для XPath 1.0 и 2.0.

Математические операторы

Набор математических операторов в XPath весьма ограничен. Следующие две таблицы стилей демонстрируют работу математических операторов; первая таблица написана для XSLT 1.0, а вторая – для XSLT 2.0. Мы подробно рассмотрим таблицы стилей для первого оператора (+), а затем просто будем ссылаться на эти примеры в других подразделах.

Сложение (+)

Оператор + суммирует два числа.

[1.0] В таблицах стилей XSLT 1.0 процессор пытается преобразовать каждый операнд в число. Все следующие выражения XPath работают в XPath 1.0:

```
<?xml version="1.0"?>
<!-- addition-1_0.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Tests of addition in XPath 1.0&#xA;</xsl:text>
    <xsl:text>&#xA;  9 + 3 = </xsl:text>
    <xsl:value-of select="9 + 3"/>
    <xsl:text>&#xA;  9 + 3.8 = </xsl:text>
    <xsl:value-of select="9 + 3.8"/>
    <xsl:text>&#xA;  9 + '4' = </xsl:text>
    <xsl:value-of select="9 + '4'"/>
    <xsl:text>&#xA;  9 + 'Q' = </xsl:text>
    <xsl:value-of select="9 + 'Q'"/>
    <xsl:text>&#xA;  9 + true() = </xsl:text>
    <xsl:value-of select="9 + true()"/>
    <xsl:text>&#xA;  9 + false() = </xsl:text>
    <xsl:value-of select="9 + false()"/>
  </xsl:template>
</xsl:stylesheet>
```

Результат:

```
Tests of addition in XPath 1.0

  9 + 3 = 12
  9 + 3.8 = 12.8
  9 + '4' = 13
  9 + 'Q' = NaN
  9 + true() = 10
  9 + false() = 9
```

Обратите внимание: в XSLT 1.0 строка (`'4'`) преобразуется в число. Если строка не может быть преобразована в число (`'Q'`), то результатом является `NaN` («нечисло»). Логические значения, возвращаемые функциями `true()` и `false()`, преобразуются в 1 и 0 соответственно.

[2.0] **В XSLT 2.0 действуют более жесткие требования. Два операнда должны относиться к совместимым типам данных. Если привести строку таблицы стилей к виду `<xsl:stylesheet version="2.0" ...>`, обработка таблицы завершится неудачей. Первые две операции (`9 + 3` и `9 + 3.8`) работают нормально, но остальные операции работать не будут. Проблему можно решить преобразованием значений в числа функцией `number()`. Обновленная таблица стилей выглядит так:**

```
<?xml version="1.0"?>
<!-- addition-2_0.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Tests of addition in XPath 2.0&#xA;</xsl:text>
    <xsl:text>&#xA;  9 + 3 = </xsl:text>
    <xsl:value-of select="9 + 3"/>
    <xsl:text>&#xA;  9 + 3.8 = </xsl:text>
    <xsl:value-of select="9 + 3.8"/>
    <xsl:text>&#xA;  9 + number('4') = </xsl:text>
    <xsl:value-of select="9 + number('4')"/>
    <xsl:text>&#xA;  9 + number('Q') = </xsl:text>
    <xsl:value-of select="9 + number('Q')"/>
    <xsl:text>&#xA;  9 + number(true()) = </xsl:text>
    <xsl:value-of select="9 + number(true())"/>
    <xsl:text>&#xA; 9 + number(false()) = </xsl:text>
    <xsl:value-of select="9 + number(false())"/>
  </xsl:template>
</xsl:stylesheet>
```

После явного преобразования каждого значения в число таблица стилей выдает те же результаты, что и таблица XSLT 1.0:

```
Tests of addition in XPath 2.0

  9 + 3 = 12
  9 + 3.8 = 12.8
  9 + number('4') = 13
  9 + number('Q') = NaN
  9 + number(true()) = 10
  9 + number(false()) = 9
```

Еще одно возможное решение – включить атрибут XSLT 2.0 `version="1.0"` в любой из элементов `<xsl:value-of>`. Следующий элемент будет работать в таблице стилей XSLT 2.0:

```
<xsl:value-of select="9 + '4'" version="1.0"/>
```

Он выдает значение 13. Атрибут `version` может быть включен в любой элемент XSLT; он сообщает, что элемент должен обрабатываться по правилам XSLT 1.0.

Если суммируемые значения являются узлами, а не атомарными значениями, динамическая типизация XPath 1.0 работает аналогичным образом. Например, в следующем элементе возраст Шекспира определяется вычитанием:

```
<xsl:value-of select="/sonnet/auth:author/year-of-death -
                      /sonnet/auth:author/year-of-birth"/>
```

Операция вычитания выполняется между узлами, поэтому оба узла сначала преобразуются в числа, а затем используются при вычитании. Попытка выполнить ту же операцию с двумя строками в XSLT 2.0 обречена на неудачу:

```
<xsl:value-of select="'1616' - '1564'"/>
```

[2.0] **В XPath 2.0 знак + может использоваться для суммирования дат, времен и временных промежутков. Возможны следующие комбинации:**

- **Два значения** `xs:yearMonthDuration`
- **Два значения** `xs:dayTimeDurations`
- `xs:yearMonthDuration` **и** `xs:dateTime`
- `xs:dayTimeDuration` **и** `xs:dateTime`
- `xs:yearMonthDuration` **и** `xs:date`
- `xs:dayTimeDuration` **и** `xs:date`
- `xs:dayTimeDuration` **и** `xs:time`.

(Обратите внимание: `xs:yearMonthDuration` **не может суммироваться с** `xs:time`**.)**

В следующей таблице стилей представлены примеры всех поддерживаемых видов суммирования:

```
<?xml version="1.0"?>
<!-- addition-datesTimesDurations.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:variable name="yMD1" as="xs:yearMonthDuration"
    select="xs:yearMonthDuration('P1Y8M')"/>
  <xsl:variable name="yMD2" as="xs:yearMonthDuration"
    select="xs:yearMonthDuration('P2Y7M')"/>
  <xsl:variable name="dTD1" as="xs:dayTimeDuration"
    select="xs:dayTimeDuration('P5DT9H23M12S')"/>
  <xsl:variable name="dTD2" as="xs:dayTimeDuration"
    select="xs:dayTimeDuration('P3DT16H12M17S')"/>
  <xsl:variable name="dT" as="xs:dateTime"
    select="xs:dateTime('1995-04-21T00:47:00')"/>
  <xsl:variable name="d" as="xs:date"
    select="xs:date('1995-04-21')"/>
  <xsl:variable name="t" as="xs:time"
    select="xs:time('17:03:00')"/>

  <xsl:template match="/">
    <xsl:text>More tests of addition in XPath 2.0:</xsl:text>
    <xsl:text>&#xA;&#xA;  Two xs:yearMonthDurations:&#xA;    </xsl:text>
    <xsl:value-of select="($yMD1, '+', $yMD2, '=', $yMD1 + $yMD2)"/>
    <xsl:text>&#xA;&#xA;  Two xs:dayTimeDurations:&#xA;    </xsl:text>
    <xsl:value-of select="($dTD1, '+', $dTD2, '=', $dTD1 + $dTD2)"/>
    <xsl:text>&#xA;&#xA;  An xs:yearMonthDuration and an </xsl:text>
    <xsl:text>xs:dateTime:&#xA;    </xsl:text>
    <xsl:value-of select="($dT, '+', $yMD1, '=', $dT + $yMD1)"/>
    <xsl:text>&#xA;&#xA;  An xs:dayTimeDuration and an </xsl:text>
    <xsl:text>xs:dateTime:&#xA;    </xsl:text>
```

```
    <xsl:value-of select="($dT, '+', $dTD1, '=', $dT + $dTD1)"/>
    <xsl:text>&#xA;&#xA;  An xs:yearMonthDuration and an </xsl:text>
    <xsl:text>xs:date:&#xA;    </xsl:text>
    <xsl:value-of select="($d, '+', $yMD1, '=', $d + $yMD1)"/>
    <xsl:text>&#xA;&#xA;  An xs:dayTimeDuration and an </xsl:text>
    <xsl:text>xs:date:&#xA;    </xsl:text>
    <xsl:value-of select="($d, '+', $dTD1, '=', $d + $dTD1)"/>
    <xsl:text>&#xA;&#xA;  An xs:dayTimeDuration and an </xsl:text>
    <xsl:text>xs:time:&#xA;    </xsl:text>
    <xsl:value-of select="($t, '+', $dTD1, '=', $t + $dTD1)"/>
  </xsl:template>
</xsl:stylesheet>
```

Результат:

```
More tests of addition in XPath 2.0:

  Two xs:yearMonthDurations:
    P1Y8M + P2Y7M = P4Y3M

  Two xs:dayTimeDurations:
    P5DT9H23M12S + P3DT16H12M17S = P9DT1H35M29S

  An xs:yearMonthDuration and an xs:dateTime:
    1995-04-21T00:47:00 + P1Y8M = 1996-12-21T00:47:00

  An xs:dayTimeDuration and an xs:dateTime:
    1995-04-21T00:47:00 + P5DT9H23M12S = 1995-04-26T10:10:12

  An xs:yearMonthDuration and an xs:date:
    1995-04-21 + P1Y8M = 1996-12-21

  An xs:dayTimeDuration and an xs:date:
    1995-04-21 + P5DT9H23M12S = 1995-04-26

  An xs:dayTimeDuration and an xs:time:
    17:03:00 + P5DT9H23M12S = 02:26:12
```

Вычитание (–)

Оператор получает два числа и вычитает второе число из первого. При использовании вычитания в приведенных ранее таблицах стилей будет получен следующий результат:

```
Tests of XPath subtraction in XSLT 1.0

  9 - 3 = 6
  9 - 3.8 = 5.2
  9 - '4' = 5
  9 - 'Q' = NaN
  9 - true() = 8
  9 - false() = 9
```

Чтобы получить те же самые результаты в таблице XSLT 2.0, следует либо преобразовать каждый из аргументов в число, либо воспользоваться атрибутом `version`.

[2.0] В XPath 2.0 знак - может использоваться для вычитания временных промежутков друг из друга, а также из значений `xs:date`, `xs:dateTime` и `xs:time`. Возможны следующие комбинации:

- Одно значение `xs:yearMonthDuration` вычитается из другого
- Одно значение `xs:dayTimeDurations` вычитается из другого
- `xs:yearMonthDuration` вычитается из `xs:dateTime`
- `xs:dayTimeDuration` вычитается из `xs:dateTime`
- `xs:yearMonthDuration` вычитается из `xs:date`
- `xs:dayTimeDuration` вычитается из `xs:date`
- `xs:dayTimeDuration` вычитается из `xs:time`.

При замене оператора + оператором - для второй таблицы стилей будет получен следующий результат:

```
More tests of subtraction in XPath 2.0:

  One xs:yearMonthDuration from another:
    P1Y8M - P2Y7M = -P11M

  One xs:dayTimeDuration from another:
    P5DT9H23M12S - P3DT16H12M17S = P1DT17H10M55S

  An xs:yearMonthDuration from an xs:dateTime:
    1995-04-21T00:47:00 - P1Y8M = 1993-08-21T00:47:00

  An xs:yearMonthDuration from an xs:dateTime:
    1995-04-21T00:47:00 - P1Y8M = 1993-08-21T00:47:00

  An xs:dayTimeDuration from an xs:dateTime:
    1995-04-21T00:47:00 - P5DT9H23M12S = 1995-04-15T15:23:48

  An xs:yearMonthDuration from an xs:date:
    1995-04-21 - P1Y8M = 1993-08-21

  An xs:dayTimeDuration from an xs:date:
    1995-04-21 - P5DT9H23M12S = 1995-04-15

  An xs:dayTimeDuration from an xs:time:
    17:03:00 - P5DT9H23M12S = 07:39:48
```

Обратите внимание: при первом вычитании получается *отрицательное* значение `-P11M`.

Умножение (*)

Оператор получает два числа и умножает первое число на второе. При использовании оператора * в первой таблице будет получен следующий результат:

```
Tests of XPath multiplication in XPath 1.0

  9 * 3 = 27
  9 * 3.8 = 34.199999999999996
  9 * '4' = 36
```

```
  9 * 'Q' = NaN
  9 * true() = 9
  9 * false() = 0
```

Результаты для XPath 2.0 выглядят более аккуратно: `34.19999...` **округляется до** `34.2`**:**

```
Tests of XPath multiplication in XPath 2.0

  9 * 3 = 27
  9 * 3.8 = 34.2
  9 * number('4') = 36
  9 * number('Q') = NaN
  9 * number(true()) = 9
  9 * number(false()) = 0
```

[2.0] **В XPath 2.0 значения** `xs:yearMonthDuration` **и** `xs:dayTimeDuration` **могут умножаться на числовые значения:**

```
<?xml version="1.0"?>
<!-- multiplication-datesTimesDurations.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:variable name="yMD1" as="xs:yearMonthDuration"
    select="xs:yearMonthDuration('P1Y8M')"/>
  <xsl:variable name="dTD1" as="xs:dayTimeDuration"
    select="xs:dayTimeDuration('P24DT08H00M00S')"/>

  <xsl:template match="/">
    <xsl:text>More tests of multiplication in XPath 2.0:</xsl:text>
    <xsl:text>&#xA;&#xA;  A xs:yearMonthDuration multiplied </xsl:text>
    <xsl:text>by a number:&#xA;    </xsl:text>
    <xsl:value-of select="($yMD1, '* 3 =', $yMD1 * 3)"/>
    <xsl:text>&#xA;&#xA;  A xs:dayTimeDuration multiplied </xsl:text>
    <xsl:text>by a number:&#xA;    </xsl:text>
    <xsl:value-of select="($dTD1, '* 10.5 =', $dTD1 * 10.5)"/>
  </xsl:template>
</xsl:stylesheet>
```

Результат:

```
More tests of multiplication in XPath 2.0:

  A xs:yearMonthDuration multiplied by a number:
    P1Y8M * 3 = P5Y

  A xs:dayTimeDuration multiplied by a number:
    P24DT8H * 10.5 = P255DT12H
```

(Формально числовое значение должно относиться к типу `xs:double`**, но XSLT 2.0 автоматически преобразует число** `3`**.)**

Деление (div)

Оператор получает два числа и делит первое число на второе. В большинстве языков программирования знак деления – косая черта (/), однако в XPath она является разделителем в путях доступа, поэтому вместо косой черты используется оператор `div`. При использовании оператора `div` в наших таблицах стилей для XPath 1.0 будет получен следующий результат:

```
Tests of XPath div in XPath 1.0

  9 div 3 = 3
  9 div 3.8 = 2.368421052631579
  9 div '4' = 2.25
  9 div 'Q' = NaN
  9 div true() = 9
  9 div false() = INF
```

В XPath 2.0 будет получен очень похожий результат:

```
Tests of XPath div in XPath 2.0

  9 div 3 = 3
  9 div 3.8 = 2.368421052631578947
  9 div number('4') = 2.25
  9 div number('Q') = NaN
  9 div number(true()) = 9
  9 div number(false()) = INF
```

Обратите внимание: выражение `9 div false()`, эквивалентное `9 div 0`, возвращает значение `INF` (бесконечность). Вопреки ожиданиям, деление на 0 не является фатальной ошибкой.

[2.0] Для временных промежутков также поддерживаются четыре разновидности деления:

- `xs:yearMonthDuration` делится на `xs:double`
- Одно значение `xs:yearMonthDuration` делится на другое
- `xs:dayTimeDuration` делится на `xs:double`
- Одно значение `xs:dayTimeDuration` делится на другое

Возьмем следующую таблицу стилей:

```
<?xml version="1.0"?>
<!-- div-datesTimesDurations.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:variable name="yMD1" as="xs:yearMonthDuration"
    select="xs:yearMonthDuration('P1Y8M')"/>
  <xsl:variable name="yMD2" as="xs:yearMonthDuration"
    select="xs:yearMonthDuration('P0Y5M')"/>
```

```
  <xsl:variable name="dTD1" as="xs:dayTimeDuration"
    select="xs:dayTimeDuration('P24DT08H00M00S')"/>
  <xsl:variable name="dTD2" as="xs:dayTimeDuration"
    select="xs:dayTimeDuration('P0DT4H00M00S')"/>

  <xsl:template match="/">
    <xsl:text>More tests of division in XPath 2.0:</xsl:text>
    <xsl:text>&#xA;&#xA;  A xs:yearMonthDuration divided </xsl:text>
    <xsl:text>by a number:&#xA;    </xsl:text>
    <xsl:value-of select="($yMD1, 'div 4 =', $yMD1 div 4)"/>
    <xsl:text>&#xA;&#xA;  One xs:yearMonthDuration divided </xsl:text>
    <xsl:text>by another:&#xA;    </xsl:text>
    <xsl:value-of select="($yMD1, 'div', $yMD2, '=', $yMD1 div $yMD2)"/>
    <xsl:text>&#xA;&#xA;  A xs:dayTimeDuration divided </xsl:text>
    <xsl:text>by a number:&#xA;    </xsl:text>
    <xsl:value-of select="($dTD1, 'div 4.5 =', $dTD1 div 4.5)"/>
    <xsl:text>&#xA;&#xA;  One xs:dayTimeDuration divided </xsl:text>
    <xsl:text>by another:&#xA;    </xsl:text>
    <xsl:value-of select="($dTD1, 'div', $dTD2, '=', $dTD1 div $dTD2)"/>
  </xsl:template>
</xsl:stylesheet>
```

Результат:

```
More tests of division in XPath 2.0:

  A xs:yearMonthDuration divided by a number:
    P1Y8M div 4 = P5M

  One xs:yearMonthDuration divided by another:
    P1Y8M div P5M = 4

  A xs:dayTimeDuration divided by a number:
    P24DT8H div 4.5 = P5DT9H46M40S

  One xs:dayTimeDuration divided by another:
    P24DT8H div PT4H = 146
```

[2.0] Целочисленное деление (idiv)

В XPath 2.0 появился оператор `idiv` для целочисленного деления. При наличии остатка от деления округление не выполняется. При замене `div` на `idiv` в предыдущем примере придется убрать пару ошибочных случаев:

```
<?xml version="1.0"?>
<!-- idiv.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Tests of idiv in XPath 2.0&#xA;</xsl:text>
    <xsl:text>&#xA;  9 idiv 3 = </xsl:text>
    <xsl:value-of select="9 idiv 3"/>
```

```
    <xsl:text>&#xA;  9 idiv 3.8 = </xsl:text>
    <xsl:value-of select="9 idiv 3.8"/>
    <xsl:text>&#xA;  9 idiv number('4') = </xsl:text>
    <xsl:value-of select="9 idiv number('4')"/>
    <!-- Приводит к фатальной ошибке                -->
    <!-- <xsl:value-of select="9 idiv number('Q')"/> -->
    <xsl:text>&#xA;  9 idiv number(true()) = </xsl:text>
    <xsl:value-of select="9 idiv number(true())"/>
    <!-- Приводит к фатальной ошибке                    -->
    <!-- <xsl:value-of select="9 idiv number(false())"/> -->
  </xsl:template>
</xsl:stylesheet>
```

Сравните эти результаты с результатами оператора `div`:

```
Tests of idiv in XPath 2.0

  9 idiv 3 = 3
  9 idiv 3.8 = 2
  9 idiv number('4') = 2
  9 idiv number(true()) = 9
```

Как видите, округление не выполняется.

Остаток от деления (mod)

Оператор `mod` возвращает остаток от деления. Результаты для XPath 1.0:

```
Tests of the mod operator in XPath 1.0

  9 mod 3 = 0
  9 mod 3.8 = 1.4000000000000004
  9 mod '4' = 1
  9 mod 'Q' = NaN
  9 mod true() = 0
  9 mod false() = NaN
```

Результаты для XPath 2.0 очень похожи:

```
Tests of the mod operator in XPath 2.0

  9 mod 3 = 0
  9 mod 3.8 = 1.4
  9 mod number('4') = 1
  9 mod number('Q') = NaN
  9 mod number(true()) = 0
  9 mod number(false()) = NaN
```

При использовании `mod` для деления на нуль возвращается значение `NaN` (нечисло). Этот же результат будет получен при делении данных, которые не могут быть преобразованы в числа (например, `'Q'`).

Оператор `mod` чаще всего используется для перебора нескольких фиксированных значений. Допустим, таблица стилей должна генерировать разные цвета фона для строк таблицы с чередованием серого и белого. Для решения этой задачи можно воспользоваться оператором `mod`:

```
<xsl:attribute name="bgcolor">
  <xsl:choose>
    <xsl:when test="position() mod 2 = 1">
      <xsl:text>white</xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>gray</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:attribute>
```

Для первого объекта значение `position()` равно 1. Остаток от деления 1 на 2 равен 1, поэтому первой строке таблицы будет назначен белый цвет фона. Для всех четных строк остаток будет равен 0; это означает, что цвет фона будет серым. Для чередования пяти фиксированных значений используются проверки вида `position() mod 5 = 1`, `position() mod 5 = 2` и т. д.

Унарный минус (–x)

Унарный минус изменяет знак операнда на противоположный. Следующая таблица стилей показывает, как он работает:

```
<?xml version="1.0"?>
<!-- unary-minus.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="x" as="xs:integer" select="xs:integer(-10)"/>

    <xsl:text>An example of the unary minus </xsl:text>
    <xsl:text>operator:&#xA;    </xsl:text>
    <xsl:value-of select="('$x =', $x, '&#xA;   -$x = ', -$x)"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
An example of the unary minus operator:
    $x = -10
   -$x =  10
```

Небольшое уточнение из спецификации Functions and Operators: если аргумент относится к типу `xs:integer` или `xs:decimal`, то результат изменения знака 0 или 0.0 равен 0 или 0.0 соответственно. Если аргумент относится к типу `xs:float` или `xs:double`, для `NaN` возвращается `NaN`, для `0.0E0` возвращается `-0.0E0`, для `-0.0E0` возвращается `0.0E0`, для `INF` возвращается `-INF`, а для `-INF` возвращается `INF`.

Унарный плюс (+)

Унарный плюс возвращает свой операнд без изменения знака. Он не изменяет операнда и включен в XPath только для полноты. Изменив таблицу стилей, использованную для унарного минуса, мы получим следующие результаты:

```
An example of the unary plus operator:
    $x = -10
   +$x = -10
```

Логические операторы

XPath также определяет несколько логических операторов. Все они достаточно тривиальны (во всяком случае в XPath 1.0), поэтому мы ограничимся их простым перечислением.

При работе с логическими выражениями в XPath следует помнить, что значения `'true'` и `'false'` представляют собой обычные строки. Если вам потребуется использовать логические значения, применяйте функции `true()` и `false()`. Если вы попытаетесь просто включить литерал `true` в выражение XPath, он будет обозначать узел с именем `true`, а это почти наверняка не то, что вам требовалось. Чтобы подчеркнуть это обстоятельство, мы будем ссылаться на логические значения при помощи функций.

Как упоминалось при описании математических операторов, при преобразовании `true()` и `false()` в число вы получите 1 и 0 соответственно. XPath определяет правила преобразования разных типов данных в логические значения. За подробностями обращайтесь к разделу «Преобразование к логическим значениям» главы 5.

Сравнение выражений

XPath 1.0 (и 2.0) определяет несколько операторов для сравнения выражений. Эти операторы перечислены ниже:

`=` *(равно)*

Получает два выражения; возвращает `true()`, если в результате вычисления обоих выражений будут получены одинаковые значения, и `false()` в противном случае.

`!=` *(не равно)*

Получает два выражения; возвращает `true()`, если в результате вычисления обоих выражений будут получены разные значения, и `false()` в противном случае.

`<` *или* `<` *(меньше)*

Получает два выражения; возвращает `true()`, если в результате вычисления первого выражения будет получено значение, меньшее, чем у второго выражения. В противном случае возвращается `false()`. Оператор «меньше» обычно записывается в виде `<`, чтобы он не путался с открывающей скобкой тегов XML.

`<=` *или* `<=` *(меньше или равно)*

Получает два выражения; возвращает `true()`, если в результате вычисления первого выражения будет получено значение, меньшее или равное, чем у второго выражения. В противном случае возвращается `false()`. Оператор «меньше или равно» обычно записывается в виде `<=`, чтобы он не путался с элементом XML с именем `=`.

`>` *или* `>` *(больше)*

Получает два выражения; возвращает `true()`, если в результате вычисления первого выражения будет получено значение, большее, чем у второго выражения. В противном случае возвращается `false()`. Оператор «больше» обычно записывается в виде `>`, чтобы он не путался с закрывающей скобкой тегов XML.

`>=` *или* `>=` *(больше или равно)*

Получает два выражения; возвращает `true()`, если в результате вычисления первого выражения будет получено значение, большее или равное, чем у второго выражения. В противном случае возвращается `false()`. Оператор «больше или равно» может записываться в виде `>=`.

`and`

Получает два выражения; возвращает `true()`, если в результате вычисления обоих выражений будут получены значения `true()`. Если хотя бы для одного выражения будет получен результат `false()`, оператор `and` возвращает `false()`.

`or`

Получает два выражения; возвращает `true()`, если в результате вычисления хотя бы одного выражения будет получено значение `true()`. Если для обоих выражений будет получен результат `false()`, оператор `or` возвращает `false()`.

Для выполнения логической операции `not` в XPath определяется функция `not()`. Ключевое слово `not` в XPath не существует. За подробностями обращайтесь к описанию функции `not()` в приложении C.

Во многих языках программирования определяются особые правила обработки логических операторов (таких как `and` и `or`). Например, многие языки требуют, чтобы первый операнд обрабатывался до второго, и если результат выражения известен после обработки первого операнда, второй операнд не обрабатывается. (Если вычисление первого операнда дает `false()`, результат выражения `and` заведомо равен `false()`; если вычисление первого операнда дает `true()`, результат выражения `or` заведомо равен `true()`.) В XPath такие правила не определяются, поэтому процессоры XSLT могут реализовать `and` и `or` так, как считают нужным.

[2.0] Сравнение атомарных значений

Как упоминалось ранее, в XPath 2.0 появилась концепция атомарных значений. Для их сравнения определяются шесть новых операторов:

`eq`

Проверяет условие «два значения равны».

`ge`

Проверяет условие «первое значение больше или равно второму».

`gt`

Проверяет условие «первое значение больше второго».

`le`

Проверяет условие «первое значение меньше или равно второму».

`lt`

Проверяет условие «первое значение меньше второго».

`ne`

Проверяет условие «два значения не равны».

Операторы `eq` и `ne` работают со следующими типами данных:

- Числовые значения (`xs:decimal`, `xs:double`, `xs:float`, `xs:integer`)
- Логические значения
- Временные промежутки (`xs:duration`, `xs:yearMonthDuration`, `xs:dayTimeDuration`)
- Дата и время (`xs:date`, `xs:time`, `xs:dateTime`)
- Компоненты дат (`xs:gYear`, `xs:gYearMonth`, `xs:gMonth`, `xs:gMonthDay`, `xs:gDay`)
- Полные имена (`xs:QName`)
- Двоичные данные (`xs:hexBinary`, `xs:base64Binary`)
- Нотации (`xs:NOTATION`)

Операторы `ge`, `gt`, `le` и `lt` поддерживают меньшее количество типов данных:

- Числовые значения (`xs:decimal`, `xs:double`, `xs:float`, `xs:integer`)
- Логические значения
- Временные промежутки (`xs:duration`, `xs:yearMonthDuration`, `xs:dayTimeDuration`)
- Дата и время (`xs:date`, `xs:time`, `xs:dateTime`)

Операторы сравнения значений несколько отличаются от общих операторов сравнения. Для сравнения значений устанавливаются более жесткие правила, так как два операнда должны относиться к одному типу. Для следующих элементов XML:

```
...
<brand>
  <name>Callebaut</name>
  <units>8203</units>
</brand>
...
```

выражение `brand[1]/units gt 10000` вызовет ошибку, потому что нетипизованное значение сравнивается с целым числом. Здесь необходимо сравнивать два целых числа, поэтому мы либо преобразуем значение элемента `<units>` в число, либо используем XML Schema для идентификации элемента как `xs:integer`. С общими операторами сравнения (`brand[1]/units > 10000`) все работает нормально. Но если вы все же хотите использовать операторы сравнения значений, выражение `xs:integer(brand[1]) gt 10000` тоже будет работать, хотя нам и приходится выполнять преобразование самостоятельно.

Общие операторы сравнения (`=`, `!=`, `<=`, `>=`, `<` и `>`) сравнивают не только значения, но и узлы, и последовательности, поэтому, скорее всего, вы будете использовать их гораздо чаще. Просто запомните, что в XPath 2.0 появились новые операторы сравнения, работающие с атомарными значениями; если вы работаете со значениями вместо узлов или последовательностей, они могут быть очень полезными.

[2.0] Сравнение последовательностей

Сравнение последовательностей работает не совсем так, как можно было бы ожидать. При сравнении последовательности со значением процессор XSLT сравнивает значение с каждым значением в последовательности. Если сравнение истинно *хотя бы для одного* значения в последовательности, операция возвращает `true`. В следующем примере используется документ XML с информацией о продажах разных сортов шоколада:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
```

```
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Таблица стилей для демонстрации сравнений с последовательностями:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- compare-sequences.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Comparing sequences with values:</xsl:text>
    <xsl:text>&#xA;  Sales figures (/report/brand/units):</xsl:text>s
    <xsl:text>&#xA;    </xsl:text>
    <xsl:value-of select="/report/brand/units" separator=", "/>
    <xsl:text>&#xA;&#xA;      /report/brand/units &gt;  27408 : </xsl:text>
    <xsl:value-of select="/report/brand/units &gt; 27408"/>
    <xsl:text>&#xA;      /report/brand/units &gt;= 27408 : </xsl:text>
    <xsl:value-of select="/report/brand/units &gt;= 27408"/>
    <xsl:text>&#xA;      /report/brand/units &lt;  8203  : </xsl:text>
    <xsl:value-of select="/report/brand/units &lt; 8203"/>
    <xsl:text>&#xA;      /report/brand/units &lt;= 8203  : </xsl:text>
    <xsl:value-of select="/report/brand/units &lt;= 8203"/>
    <xsl:text>&#xA;      /report/brand/units =  22101 : </xsl:text>
    <xsl:value-of select="/report/brand/units = 22101"/>
    <xsl:text>&#xA;      /report/brand/units =  17905 : </xsl:text>
    <xsl:value-of select="/report/brand/units = 17905"/>
    <xsl:text>&#xA;  not(/report/brand/units =  17905): </xsl:text>
    <xsl:value-of select="not(/report/brand/units = 17905)"/>
    <xsl:text>&#xA;&#xA;Comparing two sequences:</xsl:text>
    <xsl:variable name="testSequence1" as="xs:integer*"
      select="(8203, 22101, 27408, 19268, 14336)"/>
    <xsl:text>&#xA;  $testSequence1 (xs:integer*):&#xA;    </xsl:text>
    <xsl:value-of select="$testSequence1" separator=", "/>
    <xsl:text>&#xA;    $testSequence1 = /report/brand/units: </xsl:text>
    <xsl:value-of select="$testSequence1 = /report/brand/units"/>

    <xsl:variable name="testSequence2" as="xs:integer*"
      select="(19268, 17, 95, 6, 42)"/>
    <xsl:text>&#xA;&#xA;  $testSequence2 (xs:integer*):&#xA;    </xsl:text>
    <xsl:value-of select="$testSequence2" separator=", "/>
    <xsl:text>&#xA;    $testSequence2 = /report/brand/units: </xsl:text>
    <xsl:value-of select="$testSequence2 = /report/brand/units"/>
    <xsl:text>&#xA;    $testSequence2 &lt; /report/brand/units: </xsl:text>
    <xsl:value-of select="$testSequence2 &lt; /report/brand/units"/>
```

```
    <xsl:text>&#xA;    $testSequence2 &gt; /report/brand/units: </xsl:text>
    <xsl:value-of select="$testSequence2 &gt; /report/brand/units"/>

    <xsl:variable name="testSequence3" as="xs:string*"
      select="('blue', 'white', '19268')"/>
    <xsl:text>&#xA;&#xA;  $testSequence3 (xs:string*):&#xA;    </xsl:text>
    <xsl:value-of select="$testSequence3" separator=", "/>
    <xsl:text>&#xA;    $testSequence3 = /report/brand/units: </xsl:text>
    <xsl:value-of select="$testSequence3 = /report/brand/units"/>

    <xsl:variable name="testSequence4" as="xs:yearMonthDuration*"
      select="(xs:yearMonthDuration('P3Y8M'),
               xs:yearMonthDuration('P4Y8M'),
               xs:yearMonthDuration('P2Y9M'))"/>
    <xsl:text>&#xA;&#xA;  $testSequence4 (xs:yearMonthDuration*):</xsl:text>
    <xsl:text>&#xA;   (</xsl:text>
    <xsl:value-of select="$testSequence4" separator=", "/>
    <xsl:text>)&#xA;  $testSequence4 &gt; </xsl:text>
    <xsl:text>xs:yearMonthDuration('P4Y7M'): </xsl:text>
    <xsl:value-of select="$testSequence4 > xs:yearMonthDuration('P4Y7M')"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Comparing sequences with values:
  Sales figures (/report/brand/units):
    27408, 8203, 22101, 14336, 19268

      /report/brand/units >  27408 : false
      /report/brand/units >= 27408 : true
      /report/brand/units <  8203  : false
      /report/brand/units <= 8203  : true
      /report/brand/units =  22101 : true
      /report/brand/units =  17905 : false
  not(/report/brand/units =  17905): true

Comparing two sequences:
  $testSequence1 (xs:integer*):
    (8203, 22101, 27408, 19268, 14336)
  $testSequence1 = /report/brand/units: true

  $testSequence2 (xs:integer*):
    (19268, 17, 95, 6, 42)
  $testSequence2 = /report/brand/units: true
  $testSequence2 < /report/brand/units: true
  $testSequence2 > /report/brand/units: true

  $testSequence3 (xs:string*):
    (blue, white, 19268)
  $testSequence3 = /report/brand/units: true

  $testSequence4 (xs:yearMonthDuration*):
   (P3Y8M, P4Y8M, P2Y9M)
  $testSequence4 > xs:yearMonthDuration('P4Y7M'): true
```

В первой группе сравнений последовательность, соответствующая шаблону /report/brand/units, сравнивается с отдельными значениями. Сравнение /report/brand/units > 27408 дает результат false, потому что в последовательности нет значений, больших 27408. Когда мы переходим к оператору >=, результат сравнения становится истинным. При проверке равенства, если хотя бы одно значение в последовательности совпадает со сравниваемым значением, будет получен результат true. Обратите внимание также на инвертирование результата функцией not(); то же самое можно было сделать оператором !=.

В трех следующих примерах сравниваются две последовательности. В первом примере последовательность $testSequence1 сравнивается с последовательностью, соответствующей шаблону /report/brand/units. Две последовательности состоят из пяти одинаковых значений, хотя и следующих в разном порядке. Две последовательности при сравнении считаются равными, хотя для истинности сравнения достаточно, чтобы совпало всего одно значение. Сравнение последовательности из документа XML с $testSequence2 убедительно доказывает этот факт; хотя две последовательности имеют всего одно общее значение, они считаются равными. Кроме того, при сравнении $testSequence2 другими операторами мы видим, что $testSequence одновременно меньше, равна и больше последовательности, соответствующей шаблону /report/brand/units. Возможно, это выглядит неестественно, но сравнение работает именно так.

Третий пример отличается от предыдущих тем, что $testSequence3 является последовательностью значений xs:string. Так как в примере не используется XML Schema, значения из документа XML не типизованы. В двух предыдущих примерах процессор XSLT автоматически преобразует значения из документа в xs:integer, а в третьем примере они преобразуются в xs:string. Последовательность ('blue', 'white', '19268') совпадает, потому что преобразование нетипизованного значения разобранного элемента <units>19268</units> в строку создает значение '19268'.

Последний пример демонстрирует сравнение с последовательностью xs:yearMonthDuration. Мы создаем последовательность из трех значений и сравниваем ее со значением xs:yearMonthDuration('P4Y7M'). Так как одно из значений больше 4 лет 7 месяцев, результат равен true.

Возможны и более сложные операции с последовательностями. См. разделы «[2.0] Квантифицированные выражения – some и every» и «[2.0] Операторы множеств – except, intersect и union» далее в этой главе.

[2.0] Условные выражения: if, then и else

Пожалуй, логику if-then-else нельзя отнести к числу самых элегантных возможностей XSLT. Для проверки одного условия используется элемент <xsl:if>. Если же потребуется проверить несколько условий

или добавить секцию `else`, используются элементы `<xsl:choose>`, `<xsl:when>` и `<xsl:otherwise>`. (Эти элементы рассматриваются в главе 5.) В XPath 2.0 появился чрезвычайно полезный оператор `if`. Теперь логика `if-then-else` может быть реализована внутри самого выражения XPath.

Для сравнения в XSLT 1.0 логика `if` реализуется так:

```
<?xml version="1.0"?>
<!-- if-1_0.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:param name="x" select="'10'"/>

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;An example of if-then-else logic in XSLT 1.0:</xsl:text>
    <xsl:text>&#xA;&#xA;  If $x is larger than 10, print 'Big', </xsl:text>
    <xsl:text>&#xA;    otherwise print 'Little'</xsl:text>
    <xsl:text>&#xA;&#xA;           </xsl:text>
    <xsl:choose>
      <xsl:when test="$x &gt; 10">
        <xsl:text>Big</xsl:text>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>Little</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Мы проверяем значение `$x` и выводим `Big`, если оно больше 10; в противном случае выводится строка `Little`. Ничего сложного, но элемент `<xsl:choose>` занимает целых 8 строк. В XSLT 2.0 это делается гораздо проще:

```
<?xml version="1.0"?>
<!-- if-2_0.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:param name="x" select="10"/>

  <xsl:output method="text"/>

  <xsl:template match="/">
    ...
    <xsl:value-of select="if ($x &gt; 10) then 'Big' else 'Little'"/>
    ...
  </xsl:template>

</xsl:stylesheet>
```

Всего одно выражение XPath обеспечивает тот же результат.

При использовании оператора `if` необходимо помнить о двух важных деталях: проверяемое выражение должно быть заключено в круглые скобки, а ключевое слово `else` должно присутствовать *всегда*.

[2.0] Перебор последовательности – оператор for

XSLT 2.0 уделяет особое внимание работе с последовательностями, поэтому присутствие оператора для перебора всех значений последовательности выглядит вполне естественно. По аналогии с тем, как оператор `if` позволяет реализовать логику `<xsl:choose>` в выражении XPath, оператор `for` наделяет выражения XPath мощью элемента `<xsl:for-each>`. Пример:

```
<?xml version="1.0"?>
<!-- for.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:variable name="English-months" as="xs:string*"
    select="('January', 'February', 'March', 'April',
             'May', 'June', 'July', 'August',
             'September', 'October', 'November',
             'December')"/>
  <xsl:variable name="German-months" as="xs:string*"
    select="('Januar', 'Februar', 'Mдrz', 'April',
             'Mai', 'Juni', 'Juli', 'August',
             'September', 'Oktober', 'November',
             'Dezember')"/>

  <xsl:template match="/">
    <xsl:value-of
      select="for $m in ($English-months, $German-months) return
                if (starts-with($m, 'J'))
                  then concat ($m, ' starts with J!&#xA;')
                  else ''"
      separator=""/>
  </xsl:template>

</xsl:stylesheet>
```

[2.0] Квантифицированные выражения: some и every

В XPath 2.0 поддерживаются операторы `some` и `every` для проверки последовательностей. Оператор `some` возвращает `true`, если условие истинно хотя бы для одного объекта в последовательности, а оператор `every` возвращает `false`, если условие ложно хотя бы для одного объекта в последовательности. Пример использования этих двух операторов:

```
<?xml version="1.0"?>
<!-- some-every.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:variable name="English-months" as="xs:string*"
    select="('January', 'February', 'March', 'April',
             'May', 'June', 'July', 'August',
             'September', 'October', 'November',
             'December')"/>

  <xsl:template match="/">
    <xsl:text>&#xA;An example of the XPath 2.0 every and </xsl:text>
    <xsl:text>some operators:&#xA;&#xA;</xsl:text>
    <xsl:text>  If ANY month name has a string-length() </xsl:text>
    <xsl:text>&#xA;    greater than 4, print 'Yes,' otherwise</xsl:text>
    <xsl:text>&#xA;    print 'No'&#xA;&#xA;         </xsl:text>

    <xsl:value-of
      select="if (some $m in $English-months satisfies
              (string-length($m) &gt; 4)) then 'Yes' else 'No'"/>

    <xsl:text>&#xA;</xsl:text>
    <xsl:text>&#xA;&#xA;  If EVERY month name has a string-</xsl:text>
    <xsl:text>length() &#xA;    greater than 4, print 'Yes,' </xsl:text>
    <xsl:text>otherwise&#xA;    print 'No'&#xA;&#xA;         </xsl:text>

    <xsl:value-of
      select="if (every $m in $English-months satisfies
              (string-length($m) &gt; 4)) then 'Yes' else 'No'"/>

    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

К последовательности, содержащей названия месяцев на английском языке, применяются операторы `some` **и** `every`**. Мы проверяем, что длина названия каждого месяца (**`string-length()`**) больше 4. Выражение с** `some` **возвращает** `true`**, а выражение с** `every` **возвращает** `false`**. Таблица стилей написана исключительно для демонстрации новых операторов; она не использует данные из документа XML. В более привычной ситуации** `some` **и** `every` **применялись бы к выражению XPath, выбирающему узлы из документа XML. Если заменить переменную** `$English-months` **выражением** `/sonnet/lines/line`**, оба приведенных выше выражения вернут** `true` **при условии, что таблица стилей будет использоваться для обработки** `sonnet.xml`**.**

Сложности с использованием операторов some и every

Проверяемая последовательность вполне может оказаться пустой. При проверке выражения `some $m in /sonnet/lines/words` (или чего угодно, что возвращает пустую последовательность), оператор `some` вернет `false`, тогда как оператор `every` вернет `true`. Этот противоестественный на первый взгляд результат определяется в спецификации «XPath 2.0 and XQuery 1.0 Functions and Operators» (раздел 3.9, если надумаете заглянуть). Такие результаты для пустых последовательностей выглядят более разумно, если вы представите простой способ вычисления этих операторов:

- Для оператора `some` изначально предполагаем, что результат равен `false`. Затем условие проверяется для каждого объекта в последовательности, и если оно когда-либо окажется истинным, проверка прекращается, и оператор возвращает `true`. Если после перебора последовательности до конца ни один объект, для которого условие было бы истинным, так и не обнаружен, оператор возвращает `false`.
- Для оператора `every` изначально предполагаем, что результат равен `true`. Затем условие проверяется для каждого объекта в последовательности, и если оно когда-либо окажется ложным, проверка прекращается, и оператор возвращает `false`. Если после перебора последовательности до конца ни один объект, для которого условие было бы ложным, так и не обнаружен, оператор возвращает `true`.

Однако при таком подходе возникает следующая сложность: при проверке *всех* объектов последовательности может возникать фатальная ошибка. Спецификация XPath не определяет никакого обязательного поведения для таких случаев. Если процессор XSLT проверяет объекты только до тех пор, пока значение `some` или `every` не определится, ошибка может остаться необнаруженной. Два примера из спецификации XPath:

```
some $x in (1, 2, "cat") satisfies $x * 2 = 4
every $x in (1, 2, "cat") satisfies $x * 2 = 4
```

Если процессор XSLT обрабатывает объекты последовательности от первого к последнему и прекращает обработку сразу же после определения результата, фатальная ошибка `"cat" * 2` так и не будет обнаружена. Для значения 2 будет получен результат `true`, поэтому `some` возвращает `true`. Для значения 1 будет получен результат `false`, поэтому `every` возвращает `false`.

Спецификация XPath не указывает, как процессор должен реализовать `some` и `every`, и не определяет порядок обработки объектов последовательности. Возможно, из-за особенностей внутренних структур данных процессору будет проще начать с конца последовательности; в этом случае оба выражения выдадут фатальную ошибку. Самое неприятное, что фатальная ошибка может периодически возникать и исчезать в зависимости от входных данных. Чтобы предотвратить это поведение, следует использовать операторы типизации данных, которые мы вскоре рассмотрим. (Также можно проверить корректность данных до их передачи процессору XSLT – при условии, что вы доверяете своему источнику данных...)

[2.0] Диапазонные выражения – оператор to

Для создания последовательностей целых чисел в XPath 2.0 появился оператор `to`. Например, следующая короткая таблица стилей создает две последовательности из пяти целых чисел в прямом и обратном порядке, а затем выводит значения обеих последовательностей:

```
<?xml version="1.0"?>
<!-- to.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:variable name="some-numbers" as="xs:integer*"
    select="1 to 5"/>

  <xsl:variable name="reversed-numbers" as="xs:integer*"
    select="reverse(1 to 5)"/>

  <xsl:template match="/">
    <xsl:value-of select="$some-numbers" separator=", "/>
    <xsl:text>
</xsl:text>
    <xsl:value-of select="$reversed-numbers" separator=", "/>
  </xsl:template>

</xsl:stylesheet>
```

Оператор `to` создает только диапазоны целых чисел. Если второе число меньше первого (например, `10 to 1`), то результат представляет собой пустую последовательность. Если второе число равно первому (например, `10 to 10`), результат представляет собой последовательность из одного целого числа.

Оператор to может использоваться как часть большей последовательности, например (1 to 17, 65 to 100). Наконец, если вам потребуется создать последовательность, упорядоченную по убыванию, используйте функцию [2.0] reverse() (см. приложение C) для обращения (т. е. перестановки элементов в обратном порядке) последовательности, созданной оператором to.

Результат выглядит так:

```
1, 2, 3, 4, 5
5, 4, 3, 2, 1
```

[2.0] Функции-конструкторы

В XPath 2.0 появилась концепция *функций-конструкторов*, которые являются аналогами конструкторов в объектно-ориентированных языках. Как и в объектно-ориентированных языках, имя функции-конструктора совпадает с именем типа данных. Вот как создается значение типа xs:date:

```
<xsl:variable name="birthday" select="xs:date('1995-04-21')"/>
```

Функция получает строку 1995-04-21 и создает новое значение типа xs:date. Естественно, это может привести к ошибкам времени выполнения. Таблица стилей со следующей командой выполняться не будет:

```
<xsl:variable name="birthday" select="xs:date('next Tuesday')"/>
```

При выполнении произойдет статическая ошибка; процессор знает, что такую команду выполнить невозможно. С другой стороны, при передаче некорректных данных конструктору xs:date во время обработки таблицы стилей произойдет ошибка времени выполнения:

```
<xsl:variable name="birthday" select="xs:date(@birthday)"/>
```

Атрибут birthday текущего узла используется для создания нового значения xs:date. Если этот атрибут содержит значение, преобразуемое к типу xs:date, все замечательно; если содержимое атрибута некорректно, происходит ошибка времени выполнения.

[2.0] Операторы типов данных: instance of, castable as, cast as и treat as

Как и следовало ожидать, в языке, поддерживающем типы данных и конструкторы, также предусмотрены операторы для преобразования значений от одного типа к другому. XPath 2.0 поддерживает четыре таких оператора: instance of, castable as, cast as и treat as. Эти операторы рассматриваются ниже.

instance of

Оператор `instance of` проверяет, относится ли значение к конкретному типу данных. Несколько примеров:

```
<?xml version="1.0"?>
<!-- instance-of.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Some tests of the "instance of" operator:</xsl:text>

    <xsl:text>&#xA;&#xA;  '1995-04-21' instance of xs:date: </xsl:text>
    <xsl:value-of select="'1995-04-21' instance of xs:date"/>
    <xsl:text>&#xA;  xs:date('1995-04-21') instance of xs:date: </xsl:text>
    <xsl:value-of select="xs:date('1995-04-21') instance of xs:date"/>
    <xsl:text>&#xA;&#xA;  3 instance of xs:integer: </xsl:text>
    <xsl:value-of select="3 instance of xs:integer"/>
    <xsl:text>&#xA;  '3' instance of xs:integer: </xsl:text>
    <xsl:value-of select="'3' instance of xs:integer"/>
    <xsl:text>&#xA;  number('3') instance of xs:integer: </xsl:text>
    <xsl:value-of select="number('3') instance of xs:integer"/>
    <xsl:text>&#xA;  number('3') instance of xs:double: </xsl:text>
    <xsl:value-of select="number('3') instance of xs:double"/>
    <xsl:text>&#xA;  xs:integer('3') instance of xs:integer: </xsl:text>
    <xsl:value-of select="xs:integer('3') instance of xs:integer"/>
    <xsl:text>&#xA;  'e' instance of xs:integer: </xsl:text>
    <xsl:value-of select="'e' instance of xs:integer"/>

  </xsl:template>
</xsl:stylesheet>
```

Результат:

```
Some tests of the "instance of" operator:

  '1995-04-21' instance of xs:date: false
  xs:date('1995-04-21') instance of xs:date: true

  3 instance of xs:integer: true
  '3' instance of xs:integer: false
  number('3') instance of xs:integer: false
  number('3') instance of xs:double: true
  xs:integer('3') instance of xs:integer: true
  'e' instance of xs:integer: false
```

Первый пример проверяет, является ли строковое значение `1995-04-21` экземпляром типа `xs:date`. Это условие ложно; XPath не пытается автоматически преобразовать строку к `xs:date`. Во втором примере конструктор `xs:date` создает новое значение `xs:date` со строкой. На этот раз условие истинно.

Завершающие шесть примеров проверяют тип данных xs:integer. Атомарное значение 3, как и следовало ожидать, относится к типу xs:integer. Строка '3' к типу xs:integer не относится, хотя ее и можно преобразовать к нему (см. описание оператора cast as). Помимо преобразования типа, строка может быть преобразована в число функцией number(). Функция number() возвращает xs:double, поэтому instance of для xs:integer возвращает ложное значение. При использовании функции-конструктора xs:integer() возвращается, конечно же, xs:integer. Наконец, значение 'e' не относится к типу xs:integer и не может быть преобразовано к нему.

cast as

Оператор instance of проверяет тип данных значения без его фактического приведения к указанному типу. В некоторых ситуациях нужно взять значение некоторого типа данных и создать эквивалентное значение другого типа. В других языках программирования эта задача решается при помощи механизма *преобразования типа*, а в XPath 2.0 для нее определен оператор cast as. Несколько примеров:

```
<?xml version="1.0"?>
<!-- cast-as.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Some tests of the "cast as" operator:</xsl:text>

    <xsl:text>&#xA;&#xA;  '1995-04-21' cast as xs:date: </xsl:text>
    <xsl:value-of select="'1995-04-21' cast as xs:date"/>
    <xsl:text>&#xA;  '3' cast as xs:integer: </xsl:text>
    <xsl:value-of select="'3' cast as xs:integer"/>
    <xsl:text>&#xA;  3 cast as xs:integer: </xsl:text>
    <xsl:value-of select="3 cast as xs:integer"/>
    <xsl:text>&#xA;  'e' cast as xs:integer: </xsl:text>
    <xsl:text>[causes a fatal error if we try it]</xsl:text>

  </xsl:template>
</xsl:stylesheet>
```

Строка 1995-04-21 преобразуется к типу xs:date. Преобразование проходит без малейших проблем, как и два следующих. Четвертое преобразование, '3' cast as xs:integer, завершается неудачей. Таблица стилей вообще не будет работать из-за статической ошибки. Результат:

```
Some tests of the "cast as" operator:

  '1995-04-21' cast as xs:date: 1995-04-21
  '3' cast as xs:integer: 3
  3 cast as xs:integer: 3
  'e' cast as xs:integer: [causes a fatal error if we try it]
```

Естественно, при попытке извлечь значение из исходного документа XML и преобразовать его к несовместимому типу данных происходит динамическая ошибка. А если бы мы могли узнать, сработает ли преобразование типа, прежде чем применять его, то и динамической ошибки удалось бы избежать...

castable as

Оператор `castable as` проверяет, сработает ли заданное преобразование типа. Если преобразование не сработает, таблица сможет корректно исправить ошибку вместо простого сбоя. Пример использования `castable as`:

```
<?xml version="1.0"?>
<!-- castable-as.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Some tests of the "castable as" operator:</xsl:text>

    <xsl:text>&#xA;&#xA;  '1995-04-21' castable as xs:date: </xsl:text>
    <xsl:value-of select="'1995-04-21' castable as xs:date"/>
    <xsl:text>&#xA;  '3' castable as xs:integer: </xsl:text>
    <xsl:value-of select="'3' castable as xs:integer"/>
    <xsl:text>&#xA;  3 castable as xs:integer: </xsl:text>
    <xsl:value-of select="3 castable as xs:integer"/>
    <xsl:text>&#xA;  'e' castable as xs:integer: </xsl:text>
    <xsl:value-of select="'e' castable as xs:integer"/>

  </xsl:template>
</xsl:stylesheet>
```

Оператор `castable as` позволяет проверить возможность преобразования перед его выполнением. Если преобразование не сработает (скажем, `e` не преобразуется в `xs:integer`), таблица стилей сможет выполнить другое действие. Результаты обработки приведенной таблицы стилей:

```
Some tests of the "castable as" operator:

  '1995-04-21' castable as xs:date: true
  '3' castable as xs:integer: true
  3 castable as xs:integer: true
  'e' castable as xs:integer: false
```

treat as

XPath предоставляет еще один оператор, связанный с преобразованием типов: `treat as`. Чтобы понять, для чего он нужен, необходимо сначала поговорить о *статических* и *динамических типах*. Статический

тип указывается при объявлении значения. Динамический тип может быть более конкретным, чем статический, но чаще они совпадают. В спецификации XPath 2.0 приводится пример: статическим типом переменной может быть `xs:integer*`, а динамическим — `xs:integer`. Пример:

```
<xsl:variable name="integerSequence" as="xs:integer+" select="(3)"/>
```

Переменная `$integerSequence` представляет собой последовательность из одного или более целых чисел. Однако в данном экземпляре `$integerSequence` также является одиночным целым числом. В следующем примере оператор `treat as` интерпретирует последовательность целых чисел как одно целое число:

```
<?xml version="1.0"?>
<!-- treat-as.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:variable name="numbers" as="xs:integer*">
    <xsl:sequence select="/numbers/number"/>
  </xsl:variable>

  <xsl:template match="/">
    <xsl:variable name="number" as="xs:integer"
      select="$numbers treat as xs:integer"/>

    <xsl:text>&#xA;An example of the XPath 2.0 treat as </xsl:text>
    <xsl:text>operator:&#xA;&#xA;</xsl:text>
    <xsl:text>  Treat a sequence of integers as a single integer:</xsl:text>
    <xsl:text>&#xA;         </xsl:text>

    <xsl:value-of select="$number"/>
  </xsl:template>

</xsl:stylesheet>
```

Переменная состоит из последовательности значений элементов `<number>` в документе XML. Если последовательность содержит только одно число, таблица стилей работает. Если последовательность не является синглетной (если она пуста или содержит более одного целого значения), происходит ошибка времени выполнения. Например, следующий документ обрабатывается без проблем:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- numbers.xml -->
<numbers>
  <number>3</number>
</numbers>
```

Добавление элементов `<number>` (или использование документа, не содержащего ни одного элемента `<number>`) приводит к ошибке времени

выполнения. Оператор `treat as` приказывает процессору XSLT не обращать внимания на расхождения типов (`xs:integer*` и `xs:integer`), а возлагает ответственность за контроль правильности динамического типа данных на разработчика. Конечно, для этого можно воспользоваться операторами `cast as` и `castable as`.

[2.0] Операторы множеств: except, intersect и union

Одним из недостатков XPath 1.0 была невозможность сравнения наборов узлов. Если вы выбирали два набора узлов при помощи двух выражений XPath 1.0, было трудно определить, какие узлы присутствуют в обоих наборах, а какие входят только в один набор и отсутствуют в другом. В XPath 2.0 появились два новых оператора `except` и `intersect`. А для тех, кому нравится вводить лишние символы, в XPath 2.0 также был включен оператор `union` – синоним оператора `|` из XPath 1.0. (Обозначение `|` также поддерживается и в XPath 2.0.)

Учтите, что эти операторы работают только с последовательностями узлов. При попытке использовать их с последовательностями, содержащими атомарные значения, вы получите ошибку.

При рассмотрении операторов будет использоваться следующий документ XML:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- books.xml -->
<favorite-books>
  <booklist>
    <book isbn="0596000537"
      favorite="Doug Sheri">XSLT</book>
    <book isbn="0141439777"
      favorite="Doug">Tristram Shandy</book>
    <book isbn="0142437298"
      favorite="Doug">Herzog</book>
    <book isbn="0679762108"
      favorite="Doug Sheri">The Sportswriter</book>
    <book isbn="0143035479"
      favorite="Sheri">The Girls' Guide to Hunting and Fishing</book>
    <book isbn="0375724443"
      favorite="Sheri">Ava's Man</book>
  </booklist>
</favorite-books>
```

В следующих примерах выбираются две последовательности: книги, которые нравятся мне (атрибут `favorite` содержит строку `Doug`), и книги, которые нравятся моей жене (атрибут `favorite` содержит строку `Sheri`). Эти две последовательности будут использоваться для демонстрации операторов множеств.

Операторы except, intersect **и** union **сравнивают** ***сами узлы***, **а не значения этих узлов. Если два разных узла содержат одно значение, они все равно остаются разными узлами. Операторы множеств помогают выбирать разные узлы, а не разные значения. Если вас интересуют значения узлов, вероятно, стоит воспользоваться новой функцией [2.0]** distinct-values() **(см. приложение C).**

Для демонстрации оператора except **будет использоваться следующая таблица стилей:**

```
<?xml version="1.0"?>
<!-- except.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:variable name="Dougs-favorites" as="node()*">
    <xsl:sequence
      select="/favorite-books/booklist
              /book[contains(@favorite, 'Doug')]"/>
  </xsl:variable>

  <xsl:variable name="Sheris-favorites" as="node()*">
    <xsl:sequence
      select="/favorite-books/booklist
              /book[contains(@favorite, 'Sheri')]"/>
  </xsl:variable>

  <xsl:template match="/">
    <xsl:text>&#xA;Books Doug likes but Sheri doesn't:</xsl:text>
    <xsl:text>&#xA;&#xA;  </xsl:text>

    <xsl:for-each select="$Dougs-favorites except $Sheris-favorites">
      <xsl:sort select="."/>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;  </xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Другие таблицы стилей в этом разделе идентичны, но вместо except **в них используются операторы** intersect **и** union**.**

except

Оператор except **возвращает все узлы, присутствующие в первой последовательности, но не входящие во вторую. Для двух последовательностей из приведенного ранее примера выражение** $Dougs-favorites except $Sheris-favorites **дает следующий результат:**

```
Books Doug likes but Sheri doesn't:

  Herzog
  Tristram Shandy
```

intersect

Оператор `intersect` получает две последовательности узлов и возвращает последовательность из узлов, присутствующих в обеих исходных последовательностях. *Все дубликаты удаляются*; каждый узел входит в итоговую последовательность только в одном экземпляре. Результат использования оператора `intersect` в нашем примере:

```
<!-- intersect.xsl -->
...
    <xsl:for-each select="$Dougs-favorites intersect $Sheris-favorites">
...
Books we both like:

  The Sportswriter
  XSLT
```

union

Оператор `union` возвращает набор узлов, состоящих из узлов, входящих в любой из исходных наборов. Как и в случае с оператором `intersect`, дубликаты удаляются из итогового набора. Результат использования оператора `union` в нашем примере:

```
<!-- union.xsl -->
...
    <xsl:for-each select="$Dougs-favorites union $Sheris-favorites">
...
All the books we like:

  Ava's Man
  Herzog
  The Girls' Guide to Hunting and Fishing
  The Sportswriter
  Tristram Shandy
  XSLT
```

Как упоминалось ранее, оператор | (вертикальная черта) поддерживается в XPath 2.0. Строку

```
<xsl:for-each select="$Dougs-favorites union $Sheris-favorites">
```

можно заменить следующей строкой:

```
<xsl:for-each select="$Dougs-favorites | $Sheris-favorites">
```

Результат останется прежним.

Чтобы подчеркнуть тот факт, что операторы сравнивают узлы, а не их значения, мы немного изменим структуру списка книг:

```
<favorite-books>
  <favorites person="Doug">
    <book isbn="0679762108">The Sportswriter</book>
  </favorites>
```

```
  <favorites person="Sheri">
    <book isbn="0679762108">The Sportswriter</book>
  </favorites>
</favorite-books>
```

Два элемента <book> содержат одинаковые значения, но с точки зрения операторов except, intersect и union это два разных узла.

[2.0] Операторы узлов

XPath 2.0 определяет три новых оператора для работы с узлами: is, << («узел-до») и >> («узел-после»).

Оператор is

Оператор is сравнивает два узла и проверяет, совпадают ли они. Сравниваются сами узлы, а не их значения. Продолжая пример из предыдущего раздела, для демонстрации этого оператора будем использовать следующую таблицу стилей:

```
<?xml version="1.0"?>
<!-- is.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:variable name="Dougs-favorites" as="node()*">
    <xsl:sequence
      select="/favorite-books/booklist
              /book[contains(@favorite, 'Doug')]"/>
  </xsl:variable>

  <xsl:variable name="Sheris-favorites" as="node()*">
    <xsl:sequence
      select="/favorite-books/booklist
              /book[contains(@favorite, 'Sheri')]"/>
  </xsl:variable>

  <xsl:template match="/">
    <xsl:text>A test of the is operator:</xsl:text>
    <xsl:text>&#xA;  Comparing the first nodes of </xsl:text>
    <xsl:text>the sequences:&#xA;</xsl:text>

    <xsl:value-of
      select="if (subsequence($Dougs-favorites, 1, 1) is
                  subsequence($Sheris-favorites, 1, 1))
              then '    The first nodes are the same!&#xA;'
              else '    The first nodes aren''t the same!&#xA;'"/>

    <xsl:text>  Reversing one sequence and trying it </xsl:text>
    <xsl:text>again:&#xA;</xsl:text>

    <xsl:value-of
      select="if (subsequence($Dougs-favorites, 1, 1) is
                  subsequence(reverse($Sheris-favorites), 1, 1))
```

```
            then '    The first nodes are the same!&#xA;'
            else '    The first nodes aren''t the same!&#xA;'"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей создает две последовательности узлов, а затем сравнивает первые узлы каждой последовательности. В первом примере узлы совпадают. Во втором примере узлы одной из последовательностей переставляются в обратном порядке, поэтому при сравнении узлы не совпадают. (Конечно, если бы каждая последовательность содержала только один объект, результат был бы истинным.) Результат:

```
A test of the is operator:

  Comparing the first nodes of the sequences:
    The first nodes are the same!
  Reversing one sequence and trying it again:
    The first nodes aren't the same!
```

Оператор >> («узел-после»)

Оператор `>>` сравнивает два узла. Если первый узел находится после второго, `>>` возвращает `true`; в противном случае возвращается `false`. Выражение `node1 >> node1`, как и следовало ожидать, возвращает `false`; узел не может следовать после самого себя. Таблица стилей для демонстрации оператора:

```
<?xml version="1.0"?>
<!-- node-after.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:variable name="Dougs-favorites" as="node()*">
    <xsl:sequence
      select="/favorite-books/booklist
              /book[contains(@favorite, 'Doug')]"/>
  </xsl:variable>

  <xsl:template match="/">
    <xsl:text>A test of the node-after (>>) operator:</xsl:text>
    <xsl:text>&#xA;&#xA;  Comparing nodes from </xsl:text>
    <xsl:text>the sequence:&#xA;</xsl:text>

    <xsl:value-of
      select="if (subsequence($Dougs-favorites, 1, 1) >>
                  subsequence($Dougs-favorites, 2, 1))
              then '    node1 >> node2 = true&#xA;'
              else '    node1 >> node2 = false&#xA;'"/>
    <xsl:value-of
      select="if (subsequence($Dougs-favorites, 2, 1) >>
                  subsequence($Dougs-favorites, 1, 1))
              then '    node2 >> node1 = true&#xA;'
```

```
            else '    node2 >> node1 = false&#xA;'"/>
    <xsl:value-of
      select="if (subsequence($Dougs-favorites, 1, 1) >>
                  subsequence($Dougs-favorites, 1, 1))
              then '    node1 >> node1 = true&#xA;'
              else '    node1 >> node1 = false&#xA;'"/>
  </xsl:template>

</xsl:stylesheet>
```

Узлы последовательности `$Dougs-favorites` следуют в порядке документа, поэтому выражение `node1 >> node2` возвращает `false`, а `node2 >> node1` возвращает `true`. В третьем примере узел сравнивается сам с собой. Узел не может предшествовать сам себе, поэтому `node1 >> node1` возвращает `false`. Результаты:

```
A test of the node-after (>>) operator:

  Comparing nodes from the sequence:
    node1 >> node2 = false
    node2 >> node1 = true
    node1 >> node1 = false
```

Оператор << («узел-до»)

Оператор `<<` сравнивает два узла и возвращает `true`, если первый узел находится в документе до второго. Если заменить в предыдущем примере оператор `>>` оператором `<<`, будет получен следующий результат:

```
A test of the node-before (<<) operator:

  Comparing nodes from the sequence:
    node1 << node2 = true
    node2 << node1 = false
    node1 << node1 = false
```

Хотя в спецификации оператор `<<` определяется в виде `<<`, в выражении XPath знак `<` должен экранироваться. В выражениях таблицы стилей оператор `<<` используется в форме `node1 << node2`.

[2.0] Комментарии в выражениях XPath

Еще одним новшеством в синтаксисе XPath 2.0 стала возможность добавления комментариев. Синтаксис получился весьма жизнерадостным: комментарий начинается со «смайлика» `(:` и заканчивается `:)`. Пример таблицы стилей со сложной конструкцией `if`:

```
<?xml version="1.0"?>
<!-- comments.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>
```

```
  <xsl:template match="/">
    <xsl:for-each select="cars/make">
      <xsl:text>&#xA;  Car: </xsl:text>
      <xsl:value-of select="."/>
      <xsl:text> - </xsl:text>
      <xsl:value-of
        select="(: Most of our cars are from North America,
                   so we look there first :)
                if (@geography = 'North America') then
                  'Domestic car'

                (: Next, see if the car is from Europe :)
                else if (@geography = 'Europe') then
                  'Import from Europe'

                (: Check for Asia :)
                else if (@geography = 'Asia') then
                  &quot;It's from Asia&quot;

                (: If it's anything else, just say
                  'We don't know' :)
                else
                  'We don''t know!'"/>

    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Таблица содержит три команды `if`**, проверяющие значение атрибута** `geography` **элемента. Мы не поскупились на комментарии и разбивку, чтобы программа лучше читалась. Как видно из последнего комментария, экранировать апострофы в комментариях не обязательно, хотя за пределами комментариев необходимо принять соответствующие меры. Одна из выходных строк, содержащих апостроф, заключается в кавычки (**`"It's from Asia"`**). В другой строке при выводе текста внутренний апостроф удваивается (**`'We don''t know!'`**).**

Документ XML, который будет преобразован по таблице стилей:

```
<?xml version="1.0"?>
<!-- carlist-geography.xml -->
<cars>
  <make geography="Europe">Alfa Romeo</make>
  <make geography="Europe">Bentley</make>
  <make geography="North America">Chevrolet</make>
  <make geography="North America">Dodge</make>
  <make geography="North America">GMC</make>
  <make geography="Asia">Honda</make>
  <make geography="Asia">Isuzu</make>
  <make geography="?">Quantum</make>
</cars>
```

Результат:

```
Car: Alfa Romeo - Import from Europe
Car: Bentley - Import from Europe
Car: Chevrolet - Domestic car
Car: Dodge - Domestic car
Car: GMC - Domestic car
Car: Honda - Import from Asia
Car: Isuzu - Import from Asia
Car: Quantum - We don't know!
```

Эти комментарии работают только в командах XPath; они не могут использоваться для комментирования других частей таблицы стилей. Допускается также вложение комментариев. Если потребуется удалить условие `if` для европейских машин, можно закомментировать всю секцию выражения XPath:

```
(: (: Next, see if the car is from Europe :)
else if (@geography = 'Europe') then
  'Import from Europe' :)
```

Комментарий включает предыдущий комментарий и команду `if`. Комментарии XPath удобнее комментариев XML, для которых вложение запрещено.

[2.0] Типы процессоров XSLT 2.0

В спецификации XSLT 2.0 определяются две разновидности процессоров XSLT 2.0:

Схемосовместимые процессоры

Схемосовместимый процессор XSLT 2.0 поддерживает пользовательские схемы. Иначе говоря, мы можем определить собственные типы данных и структуры документа при помощи XML Schema, а затем потребовать, чтобы процессор проверил значения или узлы по этой схеме.

Базовые процессоры

Базовый процессор XSLT 2.0 поддерживает только типы данных, определенные в спецификации XML Schema Datatypes, а также несколько дополнительных типов из спецификации XPath 2.0 and XQuery 1.0 Data Model. Все эти типы данных были описаны ранее в этой главе.

За кратким введением в XML Schema и информацией об использовании схем в таблицах стилей XSLT 2.0 обращайтесь к приложению D.

Представление документа XML в XPath

В завершение темы XPath мы рассмотрим таблицу стилей, генерирующую графическое представление документа. Таблица стилей должна различать все типы узлов XPath, включая все узлы `namespace`.

Выходное представление

На рис. 3.1 показан результат применения таблицы стилей. В этом графическом представлении документа вложенные таблицы HTML показывают, какие узлы содержатся внутри других узлов, а также демонстрируют порядок следования этих узлов в исходном документе. В части документа, показанной на рис. 3.1, корень документа содержит (в указанном порядке) две инструкции по обработке и два комментария, за которыми следует элемент `<sonnet>` и еще два комментария. В свою очередь, элемент `<sonnet>` содержит два атрибута и элемент `<auth:author>`. Элемент `<auth:author>` содержит узел пространства имен и элемент.

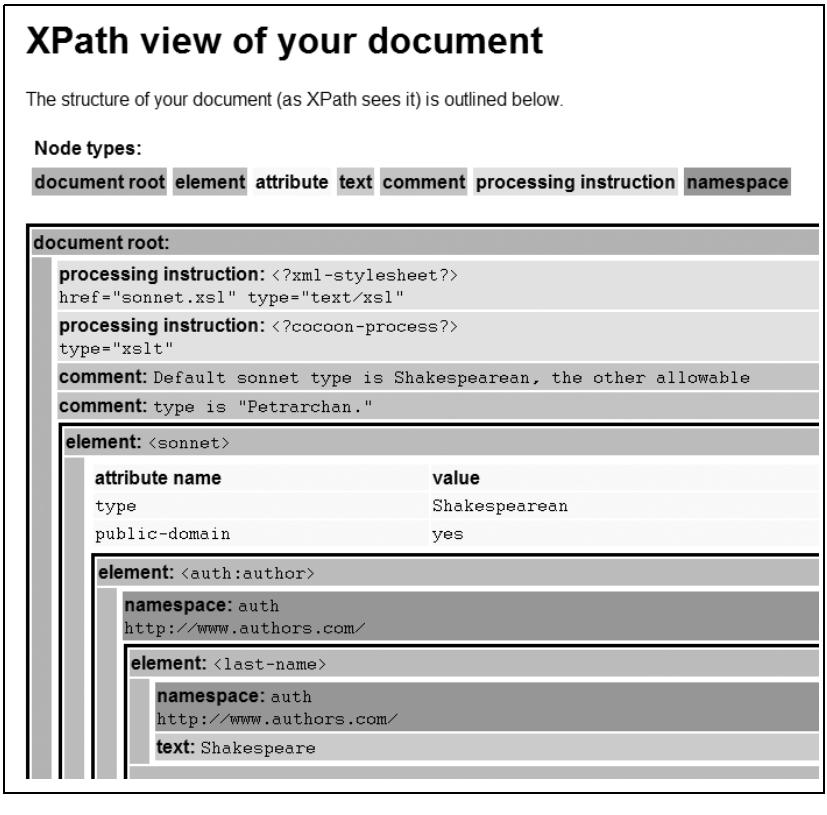


Рис. 3.1. *Представление документа XML в виде дерева XPath*

Учтите, что в случае применения таблицы стилей к очень большому документу XML вы получите файл HTML с сотнями и даже тысячами таблиц. Еще до завершения обработки документа у вашего процессора XSLT может кончиться свободная память. Например, при применении таблицы стилей для обработки исходного кода XML приложения со справочником функций создается файл HTML, содержащий более 17 000 таблиц.

Таблица стилей

Давайте посмотрим, как выглядит таблица стилей и как она работает. Таблица стилей создает серию вложенных таблиц, отражающих XPath-представление документа. Ее работа начинается с записи базовых элементов HTML в выходной поток, определения нескольких стилей CSS и создания условных обозначений для вложенного дерева. После создания условных обозначений мы выбираем все типы узлов и создаем для них представление:

```
<xsl:for-each select="*|comment()|processing-instruction()|text()">
    ...
</xsl:for-each>
```

Очень важно понимать различия между *корнем документа* XPath и *корневым элементом* XML. В примере `sonnet` за пределами корневого элемента имеются инструкции по обработке и комментарии. Корень документа, кроме собственно элемента `<sonnet>`, содержит инструкции по обработке и комментарии. В пути доступа обозначение `/` представляет корень документа, а `/sonnet` – корневой элемент документа XML. (Более общее выражение `/*` соответствует корневому элементу в любом документе.)

[2.0] Ситуация еще больше усложняется в XPath 2.0, где приходится различать тип узла (узел документа, узел элемента и т. д.) и *роль* узла: в корне документа могут находиться узлы любого типа. Только в корректном (well-formed) документе XML можно утверждать, что корневым узлом является узел документа, а его единственным дочерним элементом является элемент документа. Впрочем, при преобразованиях документов XML исключения из этого правила встречаются очень редко.

Атрибут `select` в шаблоне корня документа не включает узлы атрибутов (`@*`) и пространств имен (`namespace::*`), потому что они не могут определяться в корне документа. При попытке выбрать узлы атрибутов или пространств имен в корне документа некоторые процессоры XSLT (например, Saxon) выдают предупреждение.

Оставшаяся часть таблицы стилей содержит шаблоны для обработки всех типов узлов. В большинстве случаев обработка ограничивается выводом типа узла и его содержимого. Единственным сколько-нибудь нетривиальным шаблоном является шаблон элемента. Он создает но-

вую строку таблицы и размещает в ней вложенную таблицу. При создании новой строки используются атрибуты HTML `title` и `alt`. Каждый раз, когда пользователь наводит указатель мыши на часть таблицы, появляется всплывающее окно с информацией об иерархии элемента, представленного данной частью таблицы. Значение атрибутов `title` и `alt` создается следующим образом:

```
<xsl:variable name="title">
  <xsl:for-each select="ancestor-or-self::*">
    <xsl:text>/</xsl:text>
    <xsl:value-of select="name()"/>
  </xsl:for-each>
</xsl:variable>
<tr title="{$title}" alt="{$title}">
```

Ось `ancestor-or-self` возвращает всех предков текущего узла вместе с самим текущим узлом. Хотя ось `ancestor-or-self` возвращает узлы, находящиеся перед текущим узлом, возвращаемые узлы следуют в порядке документа. Иначе говоря, первым элементом последовательности, возвращаемой осью `ancestor-or-self`, является корневой элемент. Всплывающее окно с информацией об иерархии элемента показано на рис. 3.2.

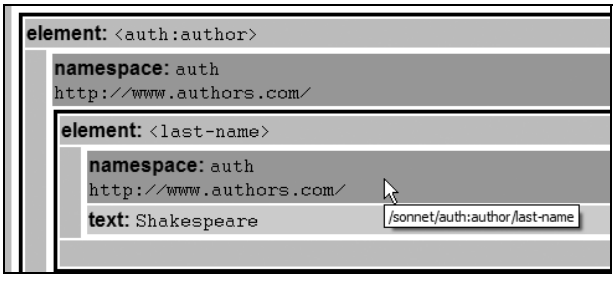


Рис. 3.2. *Всплывающее окно с информацией об иерархии элемента*

В данном примере `ancestor-or-self` возвращает последовательность узлов `<sonnet>`, `<auth-author>` и `<last-name>` (в указанном порядке).

На следующем этапе обработки элемента производится поиск определений атрибутов и пространств имен. Эти определения могут содержаться только в одном типе узлов, а именно в узлах элементов. Поиск выполняется следующим образом:

```
<table border="0" width="100%">
  <xsl:if test="count(@*) &gt; 0">
    <tr>
      <td>
        <table width="100%">
          <tr class="attribute">
            <td width="20%">
```

```
            <b>attribute name</b>
          </td>
          <td>
            <b>value</b>
          </td>
        </tr>
        <xsl:for-each select="@*">
          <tr class="attribute">
            <td width="20%">
              <span class="literal">
                <xsl:value-of select="name()"/>
              </span>
            </td>
            <td>
              <span class="literal">
                <xsl:value-of select="."/>
              </span>
            </td>
          </tr>
        </xsl:for-each>
      </table>
    </td>
  </tr>
</xsl:if>
...
```

Чтобы проиллюстрировать структуру документа, мы создаем таблицу для каждого элемента. Если элемент, в свою очередь, содержит другие элементы, они тоже размещаются в отдельных таблицах. После создания таблицы проверяется наличие узлов атрибутов (`count(@*) > 0`), и, если атрибуты присутствуют, для них тоже создается новая таблица.

После обработки всех атрибутов таблица стилей ищет узлы пространств имен и отображает их при помощи относительно редко используемой оси `namespace`:

```
<xsl:for-each select="namespace::*">
  <xsl:if test="name() != 'xml'">
    <xsl:call-template name="namespace-node"/>
  </xsl:if>
</xsl:for-each>
```

Префикс пространства имен `xml` не отображается, потому что он всегда определяется и ассоциируется с URI-адресом пространства имен `http://www.w3.org/XML/1998/namespace`. Для узлов пространств имен функция `name()` возвращает префикс пространства имен. Значение узла пространства имен (`<xsl:value-of select="."/>`) представляет собой его URI-адрес. Если бы мы искали узел пространства имен, связанный с произвольным префиксом, то проверяли бы значение узла пространства имен. Например, следующее определение пространства имен:

```
xmlns:sample="http://www.w3c.org/1999/XSL/Transform"
```

связывает пространство имен XSLT с префиксом `sample`. Чтобы узнать, определено ли пространство имен XSLT, вместо простого поиска по префиксу `xsl` нам пришлось бы использовать значение узла пространства имен.

Наконец, элемент `<xsl:apply-templates>` используется для обработки всего содержимого элемента:

```
<xsl:apply-templates select="node()"/>
```

Полный код таблицы стилей:

```
<?xml version="1.0" encoding="utf-8"?>
<!--xpath-1_0-tree-diagram.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>XPath view of your document</title>
        <style type="text/css">
          <xsl:comment>
            .literal   { font-family: Courier, monospace; }
            .docroot   { background-color: #99CCCC; }
            .element   { background-color: #CCCC99; }
            .attribute { background-color: #FFFF99; }
            .text      { background-color: #FFCC99; }
            .comment   { background-color: #CCCCFF; }
            .pi        { background-color: #99FF99; }
            .namespace { background-color: #CC99CC; }
            .box       { border: solid black 3px; }
          </xsl:comment>
        </style>
      </head>
      <body style="font-family: sans-serif;">
        <h1>XPath view of your document</h1>
        <p>
          The structure of your document (as XPath sees it)
          is outlined below.
        </p>
        <table cellspacing="5" cellpadding="2" border="0">
          <tr>
            <td colspan="7">
              <b>Node types:</b>
            </td>
          </tr>
          <tr>
            <td class="docroot"><b>document root</b></td>
            <td class="element"><b>element</b></td>
```

```
            <td class="attribute"><b>attribute</b></td>
            <td class="text"><b>text</b></td>
            <td class="comment"><b>comment</b></td>
            <td class="pi"><b>processing instruction</b></td>
            <td class="namespace"><b>namespace</b></td>
          </tr>
        </table>
        <br/>
        <table width="100%" class="box" bgcolor="#FFFFFF"
          title="document root" alt="document root">
          <tr class="docroot">
            <td colspan="3">
              <b>document root:</b>
            </td>
          </tr>
          <tr>
            <td width="15" class="docroot"></td>
            <td>
              <table width="100%">
                <xsl:apply-templates
                  select="*|comment()|processing-instruction()|text()"/>
              </table>
            </td>
            <td width="15" class="docroot"></td>
          </tr>
          <tr class="docroot">
            <td colspan="3"> </td>
          </tr>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="comment()">
    <tr>
      <td class="comment">
        <b>comment: </b>
        <span class="literal">
          <xsl:value-of select="."/>
        </span>
      </td>
    </tr>
  </xsl:template>

  <xsl:template match="processing-instruction()">
    <tr>
      <td class="pi">
        <b>processing instruction: </b>
        <span class="literal">
          <xsl:text>&lt;?</xsl:text>
          <xsl:value-of select="name()"/>
          <xsl:text>?&gt;</xsl:text>
```

```
        <br/>
        <xsl:value-of select="."/>
      </span>
    </td>
  </tr>
</xsl:template>

<xsl:template match="text()">
  <xsl:if test="string-length(normalize-space(.))">
    <tr>
      <td class="text" width="100%">
        <b>text: </b>
        <span class="literal">
          <xsl:value-of select="."/>
        </span>
      </td>
    </tr>
  </xsl:if>
</xsl:template>

<xsl:template name="namespace-node">
  <tr>
    <td class="namespace">
      <b>namespace: </b>
      <span class="literal">
        <xsl:value-of select="name()"/>
      </span>
      <br/>
      <span class="literal">
        <xsl:value-of select="."/>
      </span>
    </td>
  </tr>
</xsl:template>

<xsl:template match="*">
  <xsl:variable name="title">
    <xsl:for-each select="ancestor-or-self::*">
      <xsl:text>/</xsl:text>
      <xsl:value-of select="name()"/>
    </xsl:for-each>
  </xsl:variable>
  <tr title="{$title}" alt="{$title}">
    <td>
      <table class="box" width="100%">
        <tr>
          <td class="element" colspan="3" valign="top">
            <b>element: </b>
            <span class="literal">
              <xsl:text>&lt;</xsl:text>
              <xsl:value-of select="name()"/>
              <xsl:text>&gt;</xsl:text>
```

```
      </span>
    </td>
  </tr>
  <tr>
    <td class="element" width="15">  </td>
    <td>
      <table border="0" width="100%">
        <xsl:if test="count(@*) &gt; 0">
          <tr>
            <td>
              <table width="100%">
                <tr class="attribute">
                  <td width="20%">
                    <b>attribute name</b>
                  </td>
                  <td>
                    <b>value</b>
                  </td>
                </tr>
                <xsl:for-each select="@*">
                  <tr class="attribute">
                    <td width="20%">
                      <span class="literal">
                        <xsl:value-of select="name()"/>
                      </span>
                    </td>
                    <td>
                      <span class="literal">
                        <xsl:value-of select="."/>
                      </span>
                    </td>
                  </tr>
                </xsl:for-each>
              </table>
            </td>
          </tr>
        </xsl:if>
        <xsl:for-each select="namespace::*">
          <xsl:if test="name() != 'xml'">
            <xsl:call-template name="namespace-node"/>
          </xsl:if>
        </xsl:for-each>
        <xsl:apply-templates select="node()"/>
      </table>
    </td>
    <td bgcolor="#CCCC99" width="15"> </td>
  </tr>
  <tr>
    <td colspan="3" bgcolor="#CCCC99"> </td>
  </tr>
</table>
```

```
        </td>
      </tr>
    </xsl:template>

  </xsl:stylesheet>
```

Напоследок стоит обратить внимание на пару важных обстоятельств. Во-первых, для форматирования части вывода применяется CSS. Технологии XSLT и CSS не являются взаимоисключающими; как показывает этот пример, XSLT может использоваться для построения кода CSS в составе страницы HTML. Во-вторых, для обработки всех элементов и атрибутов в документе используются выражения с метасимволами `*` и `@*`, что позволяет применять таблицу стилей к любому документу XML независимо от используемых в нем тегов. Из-за того что в таблице стилей используются выражения с метасимволами, нам приходится получать имя элемента или атрибута, с которым мы в данный момент работаем, функцией `name()`. В-третьих, для проверки наличия атрибутов у элемента используются условная логика и выражение `count(@*) > 0`. Условная логика более подробно рассматривается в главе 5.

Итоги

В этой главе изложены основы синтаксиса XPath. Будем надеяться, что вы достаточно хорошо усвоили суть выражений XPath для описания частей документа XML. В следующих главах вам встретятся весьма разнообразные применения выражений XPath, однако все они строятся на базовых принципах, изложенных в этой главе. Как правило, в процессе отладки таблиц стилей основное время тратится именно на написание выражений XPath для выбора нужных данных. Практически все, чем мы будем заниматься в оставшейся части книги, возможно только при наличии точно сформулированных выражений XPath.

4

Создание выходных данных

К концу главы вы будете уметь:

- Генерировать текстовый вывод
- Создавать нумерацию, в том числе многоуровневую
- Форматировать числа
- *[2.0]* Форматировать даты и числа
- Использовать элементы <xsl:copy> и <xsl:copy-of> для копирования узлов из входных документов в выходной документ
- Обрабатывать пропуски

Построение текста

Начнем с выдачи текста в выходной документ. Простой вывод текстовых фрагментов выполняется достаточно легко, однако в этом разделе будут рассмотрены и некоторые нетривиальные приемы, которые еще встретятся на страницах книги. Особое внимание будет уделено двум элементам: <xsl:text> и <xsl:value-of>. С их помощью можно, например, создать документ HTML со сводкой содержимого документа XML. Для этого будем использовать следующий документ XML:

```
<?xml version="1.0"?>
<!-- toc_source.xml -->
<article>
  <title>Creating output</title>
  <body>
    <heading1>Generating text</heading1>
    <heading1>Numbering things</heading1>
    <heading1>Formatting numbers</heading1>
    <heading1>Copying nodes from the input document to the output</heading1>
    <heading1>Handling whitespace</heading1>
```

```
  </body>
</article>
```

Создание простого текста

Необходимость в записи текста в выходной документ возникает очень часто. В первом примере этой главы мы создадим фрагмент HTML-кода следующего вида:

```
<h1>Table of Contents</h1>
<h2>Generating text</h2>
<h2>Numbering things</h2>
<h2>Formatting numbers</h2>
<h2>Copying nodes from the input document to the output</h2>
<h2>Handling whitespace</h2>
```

В выходном документе текст каждого заголовка в таблице соответствует тексту некоторого элемента в исходном коде XML; общий заголовок `Table of Contents` остается неизменным. Для построения этого текста мы воспользуемся элементом `<xsl:text>`. Таблица стилей начинается с выдачи следующего текста:

```
<xsl:template match="/">
  <h1>
    <xsl:text>Table of Contents</xsl:text>
  </h1>
  ...
</xsl:template>
```

Фактически мы просто вставляем строку в выходной документ. Впрочем, код может выглядеть еще проще:

```
<xsl:template match="/">
  <h1>Table of Contents</h1>
  ...
</xsl:template>
```

По умолчанию процессор XSLT просто направляет в выходной поток все элементы таблицы стилей, не относящиеся к XSLT (например, `<h1>`). Элемент `<xsl:text>` обычно используется в тех случаях, когда необходим полный контроль над пропусками, а также при создании текстового вывода вместо документа с разметкой (например, HTML-файла).

В этих примерах используется простая запись текста в выходной документ; на практике текст чаще объединяется с некоторыми значениями из исходного документа. Например, таблица стилей может сгенерировать документ HTML следующего вида:

```
<h1>Table of Contents</h1>
<p>This document contains <b>5</b> chapters:</p>
<h2>Generating text</h2>
<h2>Numbering things</h2>
...
```

Добавленный абзац содержит как фиксированный текст, так и значение – количество глав в исходном документе. Для вывода подобных значений или названий глав используется элемент <xsl:value-of>, описание которого дается в следующем разделе.

Прежде чем переходить к элементу <xsl:value-of>, необходимо сделать небольшое замечание: если бы выходной документ генерировался на языке Java или C#, то вместо литерального текста вида Table of Contents мы использовали бы динамическую загрузку строк из файла локализации во время выполнения. Это позволило бы использовать единый код для построения сводки содержимого на английском, японском, польском или любом другом языке. Реализовать аналогичную функциональность в таблице стилей непросто, но возможно; о том, как это делается, будет рассказано в разделе «Функция document()» главы 8.

Вывод значений

Направить в документ простую текстовую строку легко: достаточно использовать элемент <xsl:text>. Тем не менее практически в каждой таблице стилей фиксированный текст объединяется со значениями из исходного документа XML. Для решения именно этой задачи и существует элемент <xsl:value-of>. В таблице стилей, над которой мы работаем, следующий элемент исходного документа XML:

```
<heading1>Generating text</heading1>
```

в выходном документе преобразуется в следующий элемент HTML:

```
<h2>Generating text</h2>
```

Иначе говоря, мы хотим взять каждый элемент <heading1> и преобразовать его в элемент HTML <h2>, содержащий значение элемента <heading1>. Шаблон для решения этой задачи выглядит так:

```
<xsl:template match="heading1">
  <h2>
    <xsl:value-of select="."/>
  </h2>
</xsl:template>
```

Приводившийся ранее абзац с количеством глав генерируется с помощью <xsl:value-of> с функцией XPath count():

```
<p>
  This document contains
  <xsl:value-of select="count(/article/body/heading1)"/>
  chapters.
</p>
```

Фрагмент HTML, созданный этим фрагментом XSLT:

```
<p>
   This document contains
   5
```

```
    chapters.
</p>
```

Как показано на рис. 4.1, броузер HTML нормализует пропуски перед выводом.

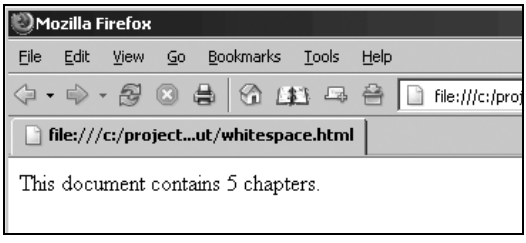


***Рис. 4.1.** HTML нормализует пропуски перед отображением документа*

Для более точного управления выводом можно совместно использовать элементы `<xsl:value-of>` и `<xsl:text>`:

```
<p>
  <xsl:text>This document contains </xsl:text>
  <xsl:value-of select="count(/article/body/heading1)"/>
  <xsl:text> chapters.</xsl:text>
</p>
```

В этом случае генерируется абзац HTML, не содержащий лишних пропусков:

```
<p>This document contains 5 chapters.</p>
```

Обратите внимание: в элемент `<xsl:text>` включаются дополнительные пробелы, чтобы число 5 отделялось от текста. Наконец, существует более простая, но хуже читаемая альтернатива: весь текст и элемент `<xsl:value-of>` размещаются в одной строке:

```
<p>This document contains <xsl:value-of select="count(/article/body...
```

Броузеры HTML обычно обрабатывают пропуски именно так, как вам требуется; для других типов вывода обработке пропусков приходится уделять особое внимание. Позже мы еще вернемся к проблеме обработки пропусков.

[2.0] Изменения в элементе <xsl:value-of> в XSLT 2.0

В XSLT 2.0 элемент `<xsl:value-of>` поддерживает атрибут `separator`. Если атрибут `select` задает последовательность объектов, эти объекты выводятся поочередно и разделяются значениями атрибута `separator`.

Возьмем короткий документ XML с перечнем фирм-автопроизводителей и выпускаемых ими машин:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- cars.xml -->
<cars>
  <manufacturer name="Chevrolet">
    <car>Cavalier</car>
    <car>Corvette</car>
    <car>Impala</car>
    <car>Malibu</car>
  </manufacturer>
  <manufacturer name="Ford">
    <car>Pinto</car>
    <car>Mustang</car>
    <car>Taurus</car>
  </manufacturer>
  <manufacturer name="Volkswagen">
    <car>Beetle</car>
    <car>Jetta</car>
    <car>Passat</car>
    <car>Touraeg</car>
  </manufacturer>
</cars>
```

Первый элемент `<xsl:value-of>` использует атрибут `separator` для вывода списка всех производителей, разделенных запятыми:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- value-of_2_0.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:value-of select="cars/manufacturer/@name" separator=", "/>
  </xsl:template>

</xsl:stylesheet>
```

Преобразование документа XML по этой таблице стилей дает следующий результат:

```
Chevrolet, Ford, Volkswagen
```

Атрибут `separator` особенно удобен в этой ситуации, поскольку он вставляется после каждого значения, кроме последнего. В XSLT 1.0 пришлось бы действовать примерно так:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- value-of_1_0.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
```

```
    <xsl:for-each select="cars/manufacturer/@name">
      <xsl:value-of select="."/>
      <xsl:if test="not(position()=last())">
        <xsl:text>, </xsl:text>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

После вывода каждого значения нам приходится проверять, является ли оно последним, и если не является – вставлять разделитель в элементе `<xsl:text>`. В XSLT 2.0 элементы `<xsl:for-each>` и `<xsl:if>` излишни; атрибут `separator` позволяет гораздо проще создать нужный нам вывод.

Атрибут `separator` также может использоваться с последовательностями. В следующем примере используется последовательность `xs:string` и последовательность, созданная новым оператором XPath 2.0 `to`:

```
<?xml version="1.0"?>
<!-- value-of_sequences.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:variable name="months" as="xs:string*"
    select="'January', 'February', 'March', 'April',
            'May', 'June', 'July', 'August',
            'September', 'October', 'November', 'December'"/>

  <xsl:template match="/">
    <xsl:value-of select="1 to 7" separator=", "/>
    <xsl:text>&#xA;</xsl:text>
    <xsl:value-of select="$months" separator="&#xA;"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей генерирует следующие результаты:

```
1, 2, 3, 4, 5, 6, 7
January
February
March
April
May
June
July
August
September
October
November
December
```

Нумерация

В XSLT существует элемент <xsl:number> для нумерации частей документа. (Он также может использоваться для форматирования числовых значений; подробнее об этом позднее.) В общем случае элемент <xsl:number> подсчитывает что-либо. В этом разделе мы рассмотрим несколько разных примеров.

Для полноценной демонстрации <xsl:number> нам понадобится документ XML с данными для подсчета. Мы снова воспользуемся списком машин из предыдущего раздела:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- cars.xml -->
<cars>
  <manufacturer name="Chevrolet">
    <car>Cavalier</car>
    <car>Corvette</car>
    <car>Impala</car>
    <car>Malibu</car>
  </manufacturer>
  <manufacturer name="Ford">
    <car>Pinto</car>
    <car>Mustang</car>
    <car>Taurus</car>
  </manufacturer>
  <manufacturer name="Volkswagen">
    <car>Beetle</car>
    <car>Jetta</car>
    <car>Passat</car>
    <car>Touraeg</car>
  </manufacturer>
</cars>
```

Несколько примеров использования <xsl:number> продемонстрируют различные возможности нумерации. Начнем с самого простого:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- number1.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Automobile manufacturers and their cars</title>
      </head>
      <body>
        <xsl:for-each select="cars/manufacturer">
          <p>
            <xsl:number format="1. "/>
```

```
            <xsl:value-of select="@name"/>
          </p>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Полученный документ HTML выглядит так:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Automobile manufacturers and their cars</title>
   </head>
   <body>
      <p>1. Chevrolet</p>
      <p>2. Ford</p>
      <p>3. Volkswagen</p>
   </body>
</html>
```

Вероятно, это самый простой пример с `<xsl:number>`, который только можно придумать. (Можно также опустить атрибут `format`, но созданные абзацы будут иметь вид `<p>1Chevrolet</p>` — вряд ли это то, что вам нужно.) Если изменить таблицу стилей так, чтобы в ней использовался атрибут `format="a. "`, сгенерированные абзацы примут следующий вид:

```
      <p>a. Chevrolet</p>
      <p>b. Ford</p>
      <p>c. Volkswagen</p>
```

А вот как выглядит результат с атрибутом `format="i. "`:

```
      <p>i. Chevrolet</p>
      <p>ii. Ford</p>
      <p>iii. Volkswagen</p>
```

Элемент `<xsl:number>` поддерживает много других атрибутов и функций; мы рассмотрим лишь самые распространенные из них. (За полным описанием элемента `<xsl:number>` обращайтесь к приложению А.) Пример использования атрибута `value`:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- number2.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Automobile manufacturers and their cars</title>
```

```
      </head>
      <body>
        <xsl:for-each select="cars/manufacturer">
          <p>
            <xsl:text>Cars produced by </xsl:text>
            <xsl:value-of select="@name"/>
            <xsl:text>: </xsl:text>
            <xsl:number value="count(car)" format="01"/>
          </p>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей генерирует следующий документ HTML:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Automobile manufacturers and their cars</title>
   </head>
   <body>
      <p>Cars produced by Chevrolet: 04</p>
      <p>Cars produced by Ford: 03</p>
      <p>Cars produced by Volkswagen: 04</p>
   </body>
</html>
```

В этом конкретном случае для получения того же результата можно было использовать элемент `<xsl:value-of select="count(car)"/>`, но `<xsl:number>` позволяет отформатировать число по нашему усмотрению.

Элемент `<xsl:number>` с атрибутом `format` удобно использовать для форматирования произвольных данных – прочитанных из источника XML или любых других. Например, следующая разметка:

```
<xsl:number value="1965" format="I"/>
```

генерирует текст `MCMLXV`.

Конечно, этим возможности элемента `<xsl:number>` не исчерпываются. В следующем примере используются атрибуты `level` и `count`:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- number3.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Automobile manufacturers and their cars&#xA;</xsl:text>
    <xsl:for-each select="cars/manufacturer">
```

```
      <xsl:number count="manufacturer" format="1. "/>
      <xsl:value-of select="@name"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:for-each select="car">
        <xsl:number count="manufacturer|car" level="multiple"
          format="1.1. "/>
        <xsl:value-of select="."/>
        <xsl:text>&#xA;</xsl:text>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей создает следующий текстовый документ:

```
Automobile manufacturers and their cars
1. Chevrolet
1.1. Cavalier
1.2. Corvette
1.3. Impala
1.4. Malibu
2. Ford
2.1. Pinto
2.2. Mustang
2.3. Taurus
3. Volkswagen
3.1. Beetle
3.2. Jetta
3.3. Passat
3.4. Touraeg
```

Атрибут `count` сообщает процессору XSLT, какие элементы следует подсчитывать, а с атрибутом `level="multiple"` фирмы и производимые ими машины нумеруются на разных уровнях. Во втором элементе `<xsl:for-each>` используется атрибут `count="manufacturer|car"`, хотя ищутся только элементы `<car>`. Это объясняется тем, что номер 3.2 означает «второй элемент `<car>` от третьего элемента `<manufacturer>`». Если не включить производителей в подсчет, мы не получим желаемого результата.

Допустимые значения атрибута `level` — `single`, `multiple` и `any`. В используемом по умолчанию режиме `single` нумеруются только одноуровневые узлы, а в режиме `multiple` при нумерации учитываются предки узла (как в приведенном примере). В режиме `level="any"` нумерация учитывает все элементы, встречавшиеся ранее в документе, независимо от того, являются они предками текущего элемента или нет. Пример использования `level="any"`:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- number4.xsl -->
<xsl:stylesheet version="1.0"
```

```
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Automobile manufacturers and their cars&#xA;</xsl:text>
    <xsl:for-each select="cars/manufacturer">
      <xsl:number count="manufacturer|car" level="any" format="1. "/>
      <xsl:value-of select="@name"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:for-each select="car">
        <xsl:number count="manufacturer|car" level="any" format="1. "/>
        <xsl:value-of select="."/>
        <xsl:text>&#xA;</xsl:text>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Здесь в обоих случаях используются идентичные элементы `<xsl:number>`. Так как подсчитываются все элементы `<manufacturer>` и `<car>`, в обоих местах задаются атрибуты `level="any"` и `count="manufacturer|car"`. Результат:

```
Automobile manufacturers and their cars
1. Chevrolet
2. Cavalier
3. Corvette
4. Impala
5. Malibu
6. Ford
7. Pinto
8. Mustang
9. Taurus
10. Volkswagen
11. Beetle
12. Jetta
13. Passat
14. Touraeg
```

Элемент `<xsl:number>` также может использоваться для выборочного подсчета. В следующем (несколько искусственном) примере нумеруются только четные машины от каждого производителя:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- number5.xsl -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Automobile manufacturers and their cars&#xA;</xsl:text>
    <xsl:for-each select="cars/manufacturer">
```

```
      <xsl:value-of select="@name"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:for-each select="car">
        <xsl:text>  </xsl:text>
        <xsl:if test="(position() mod 2) = 0">
          <xsl:number count="manufacturer|car" level="multiple"
            format="1.1. "/>
        </xsl:if>
        <xsl:value-of select="."/>
        <xsl:text>&#xA;</xsl:text>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Для выбора четных элементов применяется оператор XPath `mod`. Если остаток от деления позиции текущего элемента на 2 равен нулю, значит, текущий элемент является четным. С этой таблицей стилей мы получаем следующий список:

```
Automobile manufacturers and their cars
Chevrolet
  Cavalier
  1.2. Corvette
  Impala
  1.4. Malibu
Ford
  Pinto
  2.2. Mustang
  Taurus
Volkswagen
  Beetle
  3.2. Jetta
  Passat
  3.4. Touraeg
```

В этом примере элемент `<xsl:number>` используется только для машин с четными номерами от всех производителей, а к элементам `<manufacturer>` элемент `<xsl:number>` вообще не применяется. Несмотря на это `<xsl:number>` вычисляет правильное значение на основании позиции текущего элемента в исходном документе.

Однако при таком подходе возможны некоторые осложнения. Если исходный документ сортируется в процессе обработки, нумерация начинает выглядеть немного странно. Попробуем включить элемент `<xsl:sort>` в таблицу стилей:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- number6.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Automobile manufacturers and their cars&#xA;</xsl:text>
    <xsl:for-each select="cars/manufacturer">
      <xsl:value-of select="@name"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:for-each select="car">
        <xsl:sort select="."/>
        <xsl:text>  </xsl:text>
        <xsl:if test="(position() mod 2) = 0">
          <xsl:number count="manufacturer|car" level="multiple"
            format="1.1. "/>
        </xsl:if>
        <xsl:value-of select="."/>
        <xsl:text>&#xA;</xsl:text>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Для этой таблицы стилей будет получен следующий результат:

```
Automobile manufacturers and their cars
Chevrolet
  Cavalier
  1.2. Corvette
  Impala
  1.4. Malibu
Ford
  Mustang
  2.1. Pinto
  Taurus
Volkswagen
  Beetle
  3.2. Jetta
  Passat
  3.4. Touraeg
```

Нумерация `Pinto` не соответствует нашим ожиданиям. Дело в том, что функция `position()` работает на основе текущего (отсортированного) контекста, а нумерация осуществляется на основании исходного порядка документа. В отсортированном порядке `Pinto` стоит на втором месте, поэтому условие проверки для элемента `<xsl:if>` истинно. Но при обработке элемента `<xsl:number>` `Pinto` соответствует первому элементу `<car>` в исходном документе. Соответственно вместо ожидаемого номера 2.2 генерируется номер 2.1.

Следующая таблица стилей решает проблему, хотя такое решение изящным не назовешь. Элемент `<xsl:number>` подсчитывает элементы `<manufacturer>`; так генерируется первая часть числа. Затем позиция

отсортированного элемента выводится при помощи функции `position()`. Таблица стилей выглядит так:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- number7.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Automobile manufacturers and their cars&#xA;</xsl:text>
    <xsl:for-each select="cars/manufacturer">
      <xsl:value-of select="@name"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:for-each select="car">
        <xsl:sort select="."/>
        <xsl:text>  </xsl:text>
        <xsl:if test="(position() mod 2) = 0">
          <xsl:number count="manufacturer" level="multiple"
            format="1."/>
          <xsl:value-of select="position()"/>
          <xsl:text>. </xsl:text>
        </xsl:if>
        <xsl:value-of select="."/>
        <xsl:text>&#xA;</xsl:text>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

На этот раз нумерация `Pinto` соответствует порядку сортировки:

```
Automobile manufacturers and their cars
Chevrolet
  Cavalier
  1.2. Corvette
  Impala
  1.4. Malibu
Ford
  Mustang
  2.2. Pinto
  Taurus
Volkswagen
  Beetle
  3.2. Jetta
  Passat
  3.4. Touraeg
```

Элемент `<xsl:number>` используется для нумерации позиции текущей машины в текущем производителе.

[2.0] Изменения в элементе <xsl:number> в XSLT 2.0

В XSLT 2.0 элемент `<xsl:number>` немного изменился. Прежде всего, появилась поддержка трех новых форматов: `w`, `W` и `Ww`. Эти форматы генерируют слова на языке по умолчанию для вашего компьютера (поддерживается также атрибут `lang`, который может использоваться для смены языка). Пример:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Automobile manufacturers and their cars&#xA;</xsl:text>
    <xsl:for-each select="cars/manufacturer">
      <xsl:value-of select="@name"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:for-each select="car">
        <xsl:text>  Car </xsl:text>
        <xsl:number count="car" level="single" format="w"/>
        <xsl:text>: </xsl:text>
        <xsl:value-of select="."/>
        <xsl:text>&#xA;</xsl:text>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей генерирует следующий текст:

```
Automobile manufacturers and their cars
Chevrolet
  Car one: Cavalier
  Car two: Corvette
  Car three: Impala
  Car four: Malibu
Ford
  Car one: Pinto
  Car two: Mustang
  Car three: Taurus
Volkswagen
  Car one: Beetle
  Car two: Jetta
  Car three: Passat
  Car four: Touraeg
```

С форматом `W` сгенерированная нумерация имеет вид `ONE`, `TWO`, `THREE` и т. д., а с форматом `Ww` — `One`, `Two`, `Three` и т. д.

Еще одно отличие версии 2.0 – появление атрибута `select`. Обычно `<xsl:number>` нумерует текущий узел; атрибут `select` позволяет сгенерировать нумерацию для другого узла.

В XSLT 2.0 также добавлен новый атрибут `ordinal`. Атрибут `ordinal="yes"` в сочетании с `format="1"` генерирует нумерацию вида `1st`, `2nd` и `3rd`, тогда как с комбинацией `ordinal="yes"` с `format="Ww"` нумерация имеет вид `First`, `Second`, `Third`. Атрибут `ordinal` создает много разных вариантов нумерации в зависимости от атрибутов `lang` и `format`; как и следовало ожидать, каждый процессор XSLT поддерживает свой набор языков и возможных значений атрибута `ordinal`. За информацией об имеющихся возможностях обращайтесь к документации своего процессора.

Наконец, XSLT 2.0 выдает ошибку при наличии несовместимых атрибутов у элемента `<xsl:number>`. Например, атрибут `select` не может совмещаться с атрибутом `value`. В XSLT 1.0 лишние атрибуты игнорировались.

В приложении F приведено полное описание всех кодов форматирования чисел, даты, времени и временных интервалов.

Форматирование чисел

Мы уже видели несколько способов форматирования чисел с использованием элемента `<xsl:number>`. Но если вы собираетесь работать с числами, вам почти наверняка потребуются более совершенные средства форматирования. В XSLT для этой цели определяются элемент `<xsl:decimal-format>` и функция `format-number()`. Элемент `<xsl:decimal-format>` определяет шаблон форматирования чисел, а функция `format-number()` применяет его к числу.

Следующая таблица стилей дает несколько примеров использования `<xsl:decimal-format>` и `format-number()`:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- decimal-format.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <!-- Для формата не задано имя, поэтому он используется по умолчанию. -->
  <xsl:decimal-format decimal-separator="," grouping-separator="."/>

  <xsl:decimal-format name="us_default"/>

  <xsl:decimal-format name="other_options" NaN="[not a number]"
    infinity="unfathomably huge"/>

  <xsl:decimal-format name="hash_mark" digit="!"/>
```

```
  <xsl:template match="/">
    <xsl:text>&#xA;Tests of &lt;xsl:decimal-format&gt; and </xsl:text>
    <xsl:text>format-number():</xsl:text>
    <xsl:text>&#xA;&#xA;  1. format-number(3728493.3882, </xsl:text>
    <xsl:text>'#.###,##') : </xsl:text>
    <xsl:value-of
      select="format-number(3728493.3882, '#.###,##')"/>

    <xsl:text>&#xA;&#xA;  2. format-number(3728493.3882, </xsl:text>
    <xsl:text>'#,###.##', 'us_default') : </xsl:text>
    <xsl:value-of
      select="format-number(3728493.3882, '#,###.##', 'us_default')"/>

    <xsl:text>&#xA;&#xA;  3. format-number(number(1) div 0, '#.#') : </
xsl:text>
    <xsl:value-of select="format-number(number(1) div 0, '#.#')"/>

    <xsl:text>&#xA;&#xA;  4. format-number(number(1) div 0, '#.#', </
xsl:text>
    <xsl:text>'other_options') : &#xA;       </xsl:text>
    <xsl:value-of
      select="format-number(number(1) div 0, '#.#', 'other_options')"/>

    <xsl:text>&#xA;&#xA;  5. format-number(number('blue') * </xsl:text>
    <xsl:text>number('orange'), '#') : </xsl:text>
    <xsl:value-of
      select="format-number(number('blue') * number('orange'), '#')"/>

    <xsl:text>&#xA;&#xA;  6. format-number(number('blue') * </xsl:text>
    <xsl:text>number('orange'), '#', 'other_options') : </xsl:text>
    <xsl:text>&#xA;       </xsl:text>
    <xsl:value-of
      select="format-number(number('blue') * number('orange'), '#',
              'other_options')"/>

    <xsl:text>&#xA;&#xA;  7. format-number(42, '#!', </xsl:text>
    <xsl:text>'hash_mark') : </xsl:text>
    <xsl:value-of select="format-number(42, '#!', 'hash_mark')"/>
  </xsl:template>

</xsl:stylesheet>
```

Применение таблицы стилей к любому документу дает следующий результат:

```
Tests of <xsl:decimal-format> and format-number():

  1. format-number(3728493.3882, '#.###,##') : 3.728.493,39

  2. format-number(3728493.3882, '#,###.##', 'us_default') : 3,728,493.39

  3. format-number(number(1) div 0, '#.#') : Infinity

  4. format-number(number(1) div 0, '#.#', 'other_options') :
       unfathomably huge

  5. format-number(number('blue') * number('orange'), '#') : NaN
```

```
6. format-number(number('blue') * number('orange'), '#', 'other_options') :
     [not a number]

7. format-number(42, '#!', 'hash_mark') : #42
```

Это таблица стилей XSLT 1.0, но при изменении атрибута `version` на 2.0 результат не изменится. Мы обсудим каждую строку вывода, а заодно рассмотрим некоторые отличия механизма обработки чисел XSLT 2.0.

1. В этом примере числа форматируются с использованием определенного нами формата по умолчанию. Так как в таблице присутствует элемент `<xsl:decimal-format>` без имени, он будет использоваться по умолчанию везде, где не указано имя другого элемента `<xsl:decimal-format>`. В соответствии с нашим определением по умолчанию точка используется в качестве разделителя разрядов, а запятая отделяет дробную часть. Чтобы эта схема работала, точки и запятые должны находиться в соответствующих позициях форматной строки. В самом числовом значении для отделения дробной части как обычно используется точка.
2. Тот же пример, но с передачей имени числового формата в третьем аргументе. Обратите внимание: формат `us_default` не использует атрибутов, поэтому точка в нем используется для отделения дробной части, а запятая – в качестве разделителя групп разрядов.
3. Пример генерирует значение по умолчанию для бесконечности – строка `Infinity`. В XSLT 1.0 выражение `1 div 0` приводит к такому же результату.

   [2.0] В XSLT 2.0 значения типов `xs:integer` или `xs:decimal` не могут иметь бесконечное значение, поэтому команда `1 div 0` работать не будет. Вызов функции `number(1)` преобразует `1` в эквивалентное значение типа `xs:double`, поэтому `number(1) div 0` работает. (Деление числа на нуль требует определенных усилий, но это объясняет некоторые особенности выполнения математических вычислений в XSLT 2.0.)
4. Пример генерирует значение бесконечности, определенное в числовом формате `other_options`. Здесь действуют те же правила, что и в предыдущем примере; выражение `1 div 0` не работает в XSLT 2.0.
5. Пример генерирует значение `NaN`. В XSLT 1.0 выражение `'blue' * 'orange'` приводит к такому же результату.

   [2.0] В XSLT 2.0 выражение `'blue' * 'orange'` не работает, потому что оператор умножения работает с двумя числами. Функция `number()` преобразует каждую из строк в числовое значение `NaN`. В XSLT 2.0 значение `NaN` может быть сгенерировано вызовом `number('NaN')`, как и вызовами `number('blue')`, `number('orange')` и `number('any old string at all')`.
6. Пример генерирует `[not a number]` – значение, определенное в числовом формате `other_options`. (В XSLT 2.0 действуют те же ограничения.)

7. Пример генерирует значение #42. В нем используется числовой формат hash_mark, определяющий восклицательный знак как цифровой символ в форматной строке. Это позволило включить знак решетки (#) в выводимую строку.

[2.0] Форматирование даты и времени

В XSLT 2.0 появились три новые форматные функции: format-date(), format-time() и format-dateTime(). Мы рассмотрим несколько примеров форматирования значений xs:date, xs:time и xs:dateTime.

Каждая из этих функций может вызываться в двух вариантах. В простейшем варианте функции передаются значение и форматная строка. Если потребуется более точное управление форматированием, во втором варианте вызова функции можно задать язык, календарь и страну. В спецификации XSLT 2.0 перечислено более 25 разных календарей, используемых во всем мире, с сотнями комбинаций кодов стран и языков. За информацией о поддерживаемых календарях, языках и странах обращайтесь к документации своего процессора XSLT.

Первый пример относительно прост: мы создаем таблицу стилей с использованием функций XPath current-date(), current-time() и current-dateTime():

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datetime1.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of date and time formatting:&#xA;</xsl:text>
    <xsl:text>&#xA;  The current date is </xsl:text>
    <xsl:value-of select="format-date(current-date(),
                          '[M01]/[D01]/[Y0001]')"/>
    <xsl:text>&#xA;  The current time is </xsl:text>
    <xsl:value-of select="format-time(current-time(),
                          '[H01]:[m01] [z]')"/>
    <xsl:text>&#xA;  It's currently </xsl:text>
    <xsl:value-of select="format-dateTime(current-dateTime(),
                          '[h1]:[m01] [P] on [MNn] [D].')"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий текст:

```
Tests of date and time formatting:

  The current date is 03/08/2006
  The current time is 22:27 GMT-5
  It's currently 10:27 p.m. on March 8.
```

В этой таблице стилей формат M01 обозначает месяц из двух цифр, D01 – день из двух цифр, а Y001 – год из четырех цифр. Формат H01 обозначает час из двух цифр (по 24-часовой шкале), m01 – минуты из 2 цифр, z – часовой пояс, h1 – час из 1 или 2 цифр по 12-часовой шкале, а P – суффикс половины суток a.m. или p.m. Наконец, формат MNn генерирует название месяца с прописной буквы.

В форматных строках этих функций могут использоваться коды форматирования, поддерживаемые элементом <xsl:number>. Другая таблица стилей, в которой используется больше форматных кодов:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datetime2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;More tests of date and time formatting:&#xA;</xsl:text>
    <xsl:text>&#xA;  Today is the </xsl:text>
    <xsl:value-of select="format-date(current-date(),
                          '[Dwo] day of [MNn], [Y0001]')"/>
    <xsl:text>&#xA;  Right now is the </xsl:text>
    <xsl:value-of select="format-time(current-time(),
                          '[m1o] minute of the [Hwo] hour of the day.')"/>
    <xsl:text>&#xA;  It's currently </xsl:text>
    <xsl:value-of select="format-dateTime(current-dateTime(),
                          '[h01]:[m01] [P] on [FNn] the [D1o].')"/>
    <xsl:text>&#xA;  Today is the </xsl:text>
    <xsl:value-of select="format-date(current-date(),
                          '[dwo]')"/>
    <xsl:text> day of the year. </xsl:text>
    <xsl:text>&#xA;  December 25, 1960 in German: </xsl:text>
    <xsl:value-of select="format-date(xs:date('1960-12-25'),
                          '[D] [MNn,3-3] [Y0001]', 'de',
                          'AD', 'DE')"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей генерирует следующий текст:

```
More tests of date and time formatting:

  Today is the eighth day of March, 2006
  Right now is the 28th minute of the twenty-second hour of the day.
  It's currently 10:28 p.m. on Wednesday the 8th.
  Today is the sixty-seventh day of the year.
  December 25, 1960 in German: 25 Dez 1960
```

Далее перечислены все форматные коды этого примера:

`Dwo`

Порядковый номер дня (записанный словами).

`MNn`

Название месяца, записанное с прописной буквы.

`YOOO1`

Год из четырех цифр.

`m1o`

Числовое обозначение минут.

`Hwo`

Часы (записанные словами).

`h01`

Часы из 2 цифр (по 24-часовой шкале).

`m01`

Минуты из двух цифр.

`P`

Суффикс половины суток (`a.m.` или `p.m.`).

`FNn`

Название дня недели, записанное с прописной буквы.

`D1o`

Числовой порядковый номер дня.

`dwo`

День года (записанный словами).

`D`

День месяца (в числовом виде).

`MNn,3-3`

Название месяца, записанное с прописной буквы; возвращается в виде строки, содержащей не менее трех и не более трех символов.

При последнем вызове `format-date()` в нем используется версия функции с пятью параметрами, в которых передаются коды языка, календаря и страны. Форматные коды, определяемые XSLT, предоставляют широкий спектр вариантов форматирования компонентов даты и времени.

Элементы <xsl:copy> и <xsl:copy-of>

При преобразовании входного документа XML в другой формат часто требуется просто скопировать некоторый элемент в выходной документ. XSLT предоставляет два элемента для решения этой задачи: `<xsl:copy>` и `<xsl:copy-of>`. В этом разделе мы рассмотрим несколько таблиц стилей, в которых эти элементы используются для создания вывода.

Повторение входного документа

Начнем с таблицы стилей, которая генерирует документ, идентичный входному документу (выполняет так называемое *тождественное преобразование*). Таблица стилей получается короткой и изящной:

```
<?xml version="1.0"?>
<!-- copy-of.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <xsl:copy-of select="*"/>
  </xsl:template>
</xsl:stylesheet>
```

Вот и все, что требуется. Шаблон просто начинает с элемента документа во входном документе и копирует его в выходной документ. Элемент `<xsl:copy-of>` выполняет *глубокое копирование* узлов, поэтому корневой узел вместе со всеми потомками копируется в выходной документ. Если какой-либо из потомков корневого узла является узлом элемента с атрибутами, эти атрибуты тоже будут скопированы. (Стоит напомнить, что атрибуты элемента не являются дочерними узлами.)

Для тестирования таблицы стилей используется следующий документ:

```
<?xml version="1.0"?>
<!-- sonnet.xml -->
<sonnet type='Shakespearean'>
  <auth:author xmlns:auth="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </auth:author>
  <!-- Существует ли официальное название сонета?
       Иногда сонеты называются по первой строке. -->
  <title>Sonnet 130</title>
  <lines>
    <line>My mistress' eyes are nothing like the sun,</line>
    <line>Coral is far more red than her lips red.</line>
    <line>If snow be white, why then her breasts are dun,</line>
    <line>If hairs be wires, black wires grow on her head.</line>
    <line>I have seen roses damasked, red and white,</line>
    <line>But no such roses see I in her cheeks.</line>
    <line>And in some perfumes is there more delight</line>
    <line>Than in the breath that from my mistress reeks.</line>
    <line>I love to hear her speak, yet well I know</line>
    <line>That music hath a far more pleasing sound.</line>
    <line>I grant I never saw a goddess go,</line>
```

```
    <line>My mistress when she walks, treads on the ground.</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
  </lines>
</sonnet>
```

Результат:

```
<?xml version="1.0" encoding="UTF-8"?><!-- sonnet.xml --><sonnet type="
Shakespearean">
  <auth:author xmlns:auth="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </auth:author>
  <!-- Существует ли официальное название сонета?
       Иногда сонеты называются по первой строке. -->
  <title>Sonnet 130</title>
  <lines>
    <line>My mistress' eyes are nothing like the sun,</line>
    <line>Coral is far more red than her lips red.</line>
    <line>If snow be white, why then her breasts are dun,</line>
    <line>If hairs be wires, black wires grow on her head.</line>
    <line>I have seen roses damasked, red and white,</line>
    <line>But no such roses see I in her cheeks.</line>
    <line>And in some perfumes is there more delight</line>
    <line>Than in the breath that from my mistress reeks.</line>
    <line>I love to hear her speak, yet well I know</line>
    <line>That music hath a far more pleasing sound.</line>
    <line>I grant I never saw a goddess go,</line>
    <line>My mistress when she walks, treads on the ground.</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
  </lines>
</sonnet>
```

Итоговый документ почти не отличается от оригинала. Элемент `<sonnet>` **не начинается в отдельной строке, а в объявление XML добавлен атрибут** `encoding="UTF-8"`**. Также обратите внимание на то, что апострофы, в которые был заключен атрибут** `type`**, превратились в кавычки. Формально текст отличается от исходного документа XML, но на семантическом уровне они совпадают. Стоит особо отметить, что в выходной документ были скопированы:**

- **Комментарий перед элементом документа**
- **Атрибут** `type` **элемента** `<sonnet>`
- **Комментарий в середине документа**
- **Объявление пространства имен из элемента** `<auth:author>`

Ранее мы уже обсуждали различия между корневым узлом и элементом документа. Корневой узел в данном случае содержит элемент документа и находящийся за его пределами комментарий. Все остальные элементы исходного документа XML являются потомками корневого узла; элемент документа не всегда является единственным дочерним узлом корневого узла.

В следующем разделе читатель познакомится с элементом `<xsl:copy>` и увидит, как этот элемент работает (или не работает) с выходным документом.

Неполное повторение входного документа

Следующая таблица стилей похожа на предыдущую, но в ней используется элемент `<xsl:copy>`. Она тоже очень проста:

```
<?xml version="1.0"?>
<!-- copy1.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <xsl:copy/>
  </xsl:template>
</xsl:stylesheet>
```

Вероятно, вы заметили, что эта таблица стилей короче предыдущей. В отличие от `<xsl:copy-of>`, элемент `<xsl:copy>` не имеет атрибута `select`. (В XSLT 2.0 он имеет другие атрибуты, а в XSLT 1.0 атрибуты отсутствуют.) Результат применения этой таблицы стилей:

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

Хм... Похоже, элемент `<xsl:copy>` ничего не скопировал? Конечно, документ получился лаконичным и выразительным, но это не совсем то, чего мы хотели. (Приведенный результат получен в Xalan; Saxon вообще ничего не выдает.) Вспомните, что корень документа *не всегда* совпадает с корневым элементом, содержащим данные XML нашего документа. Документ XML может содержать комментарии и инструкции по обработке, находящиеся вне корневого элемента; они являются частью корня документа XPath. Следовательно, если мы хотим что-то скопировать, необходимо создать шаблон для корневого элемента. Вторая попытка:

```
<?xml version="1.0"?>
<!-- copy2.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <xsl:apply-templates select="*"/>
  </xsl:template>
```

```
  <xsl:template match="*">
    <xsl:copy/>
  </xsl:template>
</xsl:stylesheet>
```

А вот полученный результат:

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

По крайней мере в выходном документе появился элемент <sonnet>, но без дочерних элементов. Кроме того, потерялся атрибут type элемента <sonnet>. Если вы используете <xsl:copy> для копирования документа, придется включить элемент <xsl:for-each> для копирования всех атрибутов каждого копируемого элемента. Последняя версия таблицы стилей выглядит так:

```
<?xml version="1.0"?>
<!-- copy3.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <xsl:apply-templates select="*"/>
  </xsl:template>

  <xsl:template match="*">
    <xsl:copy>
      <xsl:for-each select="@*">
        <xsl:copy/>
      </xsl:for-each>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Элемент <xsl:for-each> копирует все атрибуты, которыми может обладать элемент. Эта таблица стилей довольно точно воспроизводит входной документ:

```
<?xml version="1.0" encoding="UTF-8"?><sonnet type="Shakespearean">
  <auth:author xmlns:auth="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </auth:author>

  <title>Sonnet 130</title>
  <lines>
    <line>My mistress' eyes are nothing like the sun,</line>
    <line>Coral is far more red than her lips red.</line>
    <line>If snow be white, why then her breasts are dun,</line>
    <line>If hairs be wires, black wires grow on her head.</line>
```

```
    <line>I have seen roses damasked, red and white,</line>
    <line>But no such roses see I in her cheeks.</line>
    <line>And in some perfumes is there more delight</line>
    <line>Than in the breath that from my mistress reeks.</line>
    <line>I love to hear her speak, yet well I know</line>
    <line>That music hath a far more pleasing sound.</line>
    <line>I grant I never saw a goddess go,</line>
    <line>My mistress when she walks, treads on the ground.</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
  </lines>
</sonnet>
```

Но даже эта версия таблицы стилей не копирует комментарии и инструкции по обработке, а комментарий в исходном документе заменился пустой строкой. Если мы хотим обрабатывать комментарии и инструкции по обработке, придется внести соответствующие изменения в атрибуты `match` и `select`.

Этими примерами я хотел показать, что с `<xsl:copy>` *большую часть работы приходится выполнять самостоятельно*. Копируемые данные находятся под вашим полным контролем, но за это приходится расплачиваться. Если вам потребуется скопировать только часть элементов по определенному признаку (например, все элементы `<customer>` с элементом `<address>`, в котором элемент `<province>` имеет значение `PEI`), `<xsl:copy>` предоставит вам такую возможность.

Прежде чем переходить к более сложным примерам использования `<xsl:copy>`, стоит упомянуть, что в этом примере также можно было воспользоваться тестом XPath `node()`. Таблица стилей

```
<?xml version="1.0"?>
<!-- copy-identity.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="node()|@*">
    <xsl:copy>
      <xsl:apply-templates select="node()|@*"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

приводит к тому же результату, что и рассмотренная ранее таблица с `<xsl:copy>`.

Рассмотрим пример, в котором полный контроль над копируемыми данными действительно необходим: следующая таблица стилей копирует только информацию `<author>` и название сонета. В ней используется отдельный шаблон, который предотвращает копирование элементов `<lines>` с текстом сонета:

```
<?xml version="1.0"?>
<!-- copy4.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <xsl:apply-templates select="*"/>
  </xsl:template>

  <xsl:template match="*">
    <xsl:copy>
      <xsl:for-each select="@*">
        <xsl:copy/>
      </xsl:for-each>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="lines"/>

</xsl:stylesheet>
```

Шаблон для элемента <lines> пуст; раз он ничего не содержит, то и не генерирует никакие выходных данные. А это означает, что вывод должен содержать все данные исходного документа, кроме элемента <lines> и его потомков. Результат выглядит так:

```
<?xml version="1.0" encoding="UTF-8"?><sonnet type="Shakespearean">
  <auth:author xmlns:auth="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </auth:author>

  <title>Sonnet 130</title>

</sonnet>
```

Выходной документ содержит всю информацию <auth:author> и <title> с некоторыми дополнительными пропусками. (Если вы захотите удалить эти пропуски, измените таблицу стилей.) С <xsl:copy-of> мы не могли этого сделать; элемент <xsl:copy> предоставляет полный контроль над выполняемыми действиями. Даже если включить в схему <sonnet> новые элементы, эта таблица стилей все равно будет копировать все, кроме элементов <lines> и их потомков.

В последней таблице стилей используется пустой шаблон <xsl:template> для элементов <lines>. Таким образом, в выходной документ копируются все остальные документы, текст и атрибуты; таблица стилей определяет, что *не должно* копироваться в выходной документ. Создать таблицу стилей, которая определяет, что *должно* копироваться, несколько сложнее:

```
<?xml version="1.0"?>
<!-- copy5.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:auth="http://www.authors.com/">

  <xsl:template match="sonnet">
    <xsl:copy>
      <xsl:copy-of select="@*"/>
      <xsl:apply-templates select="auth:author|title"/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*"/>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Здесь возникает сразу несколько затруднений. Во-первых, чтобы указать, что мы хотим скопировать уточненный пространством имен элемент <auth:author>, нам приходится определять это пространство имен в таблице стилей. Далее в элементе <sonnet> явно перечисляются два дочерних элемента. В конце таблицы стилей создается обобщенный шаблон для копирования всех обработанных элементов. По встроенным правилам шаблонов элементы <auth:author> и <title> *вместе со всеми потомками* обрабатываются обобщенным шаблоном. Элемент <lines> не обрабатывается, поэтому мы получаем желаемый результат:

```
<?xml version="1.0" encoding="UTF-8"?><sonnet type="Shakespearean"
><auth:author
xmlns:auth="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </auth:author><title>Sonnet 130</title></sonnet>
```

Мы рассмотрели два элемента XSLT, используемые для копирования: <xsl:copy-of> и <xsl:copy>. Используйте <xsl:copy-of> для глубокого копирования элемента вместе со всеми дочерними элементами и атрибутами. С другой стороны, <xsl:copy> требует явно выбрать копируемую информацию. Если вам необходим полный контроль такого рода, применяйте <xsl:copy>; во всех остальных случаях используйте <xsl:copy-of>.

Обработка пропусков

Одной из проблем при работе с любым документом XML является обработка пропусков, особенно при генерировании выходных данных в формате, отличном от HTML. Как упоминалось ранее, в HTML-броузере следующие два абзаца отображаются одинаково:

```
<p>
   This document contains
   5
   chapters.
</p>
<p>This document contains 5 chapters.</p>
```

Но если сгенерировать эти два абзаца в текстовом формате, на печати (как и в исходном коде HTML) они будут выглядеть по-разному. В этом разделе рассматриваются некоторые приемы управления пропусками.

Общие сведения о пропусках

Прежде чем переходить к делу, стоит указать, какие четыре символа в спецификации XML относятся к категории пропусков (whitespace):

- Символ *табуляции* (`	`)
- Символ *новой строки* (`
`)
- Символ *возврата курсора* (`
`)
- *Пробел* (` `)

Для демонстрации работы с пропусками в парсерах XML и процессорах XSLT будет использоваться измененная версия списка машин:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- carlist_whitespace.xml -->
<cars>
  <manufacturer name="          Chevrolet
">
    <car>Cavalier</car>
    <car>Corvette</car>
    <car>Impala</car>
    <car>Monte


Carlo</car>
  </manufacturer>
</cars>
```

С точки зрения парсера XML в этом документе несколько узлов, не содержащих ничего, кроме пропусков. Элемент `<cars>` содержит узел с символом новой строки и табуляцией или пробелами перед тегом `<manufacturer>`, узел элемента `<manufacturer>` и узел с символом новой

строки после тега </manufacturer>. Аналогичным образом элемент <manufacturer> содержит узлы, состоящие из одних пропусков, между элементами <car>.

Парсер XML не удаляет узлы, состоящие из одних пропусков, поэтому мы всегда можем использовать их в своих таблицах стилей. Иначе говоря, модель данных, используемая процессором XSLT, сохраняет узлы, состоящие из одних пропусков, присутствующие в исходном документе XML. Единственным исключением является процессор XSLT 2.0, проверяющий исходный документ XML по схеме. Если схема показывает, что элемент может содержать только другие элементы, все узлы, состоящие из одних пропусков, удаляются из таких элементов.

Процессор XSLT не знает, проверил ли парсер XML действительность документа или нет, поэтому он предполагает, что все полученные от парсера XML узлы, состоящие из одних пропусков, должны быть значимыми. Мы воспользуемся примером таблицы стилей для элемента <xsl:copy-of>, чтобы увидеть, какая информация будет получена из документа. В результате обработки короткого списка машин со следующей таблицей стилей:

```
<?xml version="1.0"?>
<!-- copy-of.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <xsl:copy-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

будет получен такой документ XML:

```
<?xml version="1.0" encoding="UTF-8"?><!-- carlist_whitespace.xml --><cars>
  <manufacturer name="          Chevrolet  ">
    <car>Cavalier</car>
    <car>Corvette</car>
    <car>Impala</car>
    <car>Monte


Carlo</car>
  </manufacturer>
</cars>
```

В преобразованной версии документа имеется только одно отличие: символы новой строки были заменены пробелами в значении атрибута name элемента <manufacturer>. Все узлы, состоящие из одних пропусков, сохранились.

Элементы <xsl:preserve-space> и <xsl:strip-space>

Спецификация XSLT определяет два режима обработки узлов, состоящих из одних пропусков. Элементы <xsl:preserve-space> и <xsl:strip-space> соответственно сохраняют и удаляют пропуски. Слегка измененная версия таблицы стилей:

```
<?xml version="1.0"?>
<!-- copy-of-whitespace.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:preserve-space elements="manufacturer"/>
  <xsl:strip-space elements="cars"/>

  <xsl:template match="/">
    <xsl:copy-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

А вот как выглядит результат обработки списка машин с измененной таблицей стилей:

```
<?xml version="1.0" encoding="UTF-8"?><!-- carlist_whitespace.xml --><cars>
        <manufacturer name="          Chevrolet  ">
    <car>Cavalier</car>
    <car>Corvette</car>
    <car>Impala</car>
    <car>Monte


Carlo</car>
  </manufacturer></cars>
```

В полученном документе узлы, состоящие из одних пропусков, были удалены из элемента <cars>, а все пропуски в элементе <manufacturer> сохранились. (Полный текст элемента <manufacturer> превышает ширину страницы.) Элементы <xsl:preserve-space> и <strip-space> могут использоваться с метасимволами:

```
<xsl:preserve-space elements="cars manufacturer"/>
<xsl:strip-space elements="*"/>
```

Эти элементы приказывают процессору XSLT удалить узлы, состоящие из одних пропусков, во всех элементах, кроме <cars> и <manufacturer>. В атрибуте elements передается звездочка или список имен элементов, разделенных пробелами.

Так как узлы, состоящие из одних пропусков, всегда сохраняются, использовать элемент вида <xsl:preserve-space elements="*"/> необязательно. Элемент <xsl:strip-space elements="*"/>, как в приведенном примере, означает, что узлы, состоящие из одних пропусков, удаляются по умолчанию. Следует явно указать элементы со смешанным содержимым, в которых пропуски важны, но при этом по возможности

удалить узлы из одних пропусков, чтобы уменьшить размер дерева узлов для больших документов с большим количеством пропусков.

Метасимволы также могут использоваться для элементов с указанием пространств имен. Например, запись `elements="auth:*"` обозначает все элементы в пространстве имен `auth`. *[2.0]* В XSLT 2.0 можно также использовать запись `elements="*:car"` для обозначения всех элементов `<car>` независимо от их пространств имен.

Элементы `<xsl:preserve-space>` и `<strip-space>` определяют, какие узлы из одних пропусков должны обрабатываться таблицей стилей. Чтобы скопировать пропуски прямо в выходной документ, используйте элементы `<xsl:copy-of>` и `<xsl:copy>`, как упоминалось выше.

И последнее замечание: в спецификации XML определяется редко используемый атрибут `xml:space`. Если элемент в исходном документе XML содержит атрибут `xml:space="preserve"`, все пропуски в этом элементе сохраняются независимо от действия каких-либо элементов `<xsl:strip-space>` в таблице стилей.

Функция normalize-space()

Другое полезное средство управления пропусками – функция `normalize-space()`. В предыдущем примере элементы `<xsl:preserve-space>` и `<xsl:strip-space>` использовались для управления узлами пропусков в различных элементах, но в атрибуте `name` и в последнем элементе `<car>` все еще остались заметные пропуски. Для их удаления можно воспользоваться функцией `normalize-space()`. Функция выполняет три операции:

- Удаление всех начальных пробелов
- Удаление всех завершающих пробелов
- Замена групп смежных пропусков одним пробелом

Следующая таблица стилей демонстрирует использование функции `normalize-space()`:

```
<?xml version="1.0"?>
<!-- normalize-space.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <xsl:apply-templates />
  </xsl:template>

  <xsl:template match="*">
    <xsl:copy>
      <xsl:for-each select="@*">
        <xsl:attribute name="{name()}">
          <xsl:value-of select="normalize-space()"/>
        </xsl:attribute>
```

```
      </xsl:for-each>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="text()">
    <xsl:value-of select="normalize-space()"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей генерирует следующий – пожалуй, даже слишком компактный – результат:

```
<?xml version="1.0" encoding="UTF-8"?><cars><manufacturer name="Chevr
olet"><car>Cavalier</car><car>Corvette</car><car>Impala</car><car>Mon
te Carlo</car></manufacturer></cars>
```

Все дополнительные пропуски в атрибуте `name` и элементе `<car>` удаляются; кроме того, функция фактически удаляет узлы, состоящие из одних пропусков.

Простой способ включения пропусков в текстовый вывод

Как правило, при выдаче любых текстовых данных нам приходится управлять разрывами строк. В языках программирования предусмотрены соответствующие средства; например, в Java функции `System.out.print()`, `System.out.println()` и даже `System.out.print("\n\n")` позволяют включить разрывы строк именно там, где они нужны.

В таблицах стилей эта задача проще всего решается при помощи символьных сущностей для символов новой строки (`
`) и табуляции (`	`). Мы уже использовали этот способ для символа новой строки в этой главе. В следующем примере таблица стилей использует все три сущности в атрибуте `separator` элемента `<xsl:value-of>`:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <xsl:text>&#xA;Values separated with newlines:&#xA;&#xA;</xsl:text>
    <xsl:value-of select="1 to 7" separator="&#xA;"/>

    <xsl:text>&#xA;&#xA;Values separated with tabs:&#xA;&#xA;</xsl:text>
    <xsl:value-of select="1 to 7" separator="&#x9;"/>

    <xsl:text>&#xA;&#xA;Values separated with spaces:&#xA;&#xA;</xsl:text>
    <xsl:value-of select="1 to 7" separator="&#x20;"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующие результаты:

```
Values separated with newlines:

1
2
3
4
5
6
7

Values separated with tabs:

1       2       3       4       5       6       7

Values separated with spaces:

1 2 3 4 5 6 7
```

Итоги

В этой главе рассматриваются различные способы генерирования вывода в таблицах стилей XSLT. Скорее всего, эти базовые приемы будут использоваться во всех написанных вами таблицах стилей. Основной проблемой, с которой мы столкнемся по мере изложения материала, станет умение выбирать и упорядочивать обрабатываемые элементы. Возможно, вам потребуется отобрать элементы, обладающие определенными свойствами, или организовать сортировку или группировку элементов перед их обработкой. А может быть, потребуется использовать специализированные функции, не определяемые в XSLT или XPath, или направить вывод не в один, а в несколько документов. Все эти темы будут рассмотрены в дальнейших главах книги.

Но что бы ни происходило в вашей таблице стилей, она все равно не обойдется без средств вывода, описанных в этой главе.

5

Программная логика и управляющие элементы

До настоящего момента мы занимались относительно тривиальными преобразованиями, правда, даже они позволяли решать довольно сложные задачи. Но для выполнения действительно полезной работы в таблицах стилей необходимо все-таки использовать программную логику. В этой главе рассматриваются элементы XSLT, предназначенные для этой цели. Некоторые из представленных элементов XML напоминают конструкции из других языков программирования, но не эквивалентны им. В процессе изложения материала мы обсудим специфику XSLT и способы выполнения стандартных операций в таблицах стилей.

К концу главы вы будете:

- Знать элементы XSLT, используемые при реализации программной логики и передачи управления
- Понимать различия между управляющими элементами XSLT и аналогичными конструкциями в других языках программирования
- Уметь вызывать шаблоны XSLT по имени и передавать им параметры (если потребуется)
- Уметь использовать переменные XSLT
- Понимать, как изменения XSLT 2.0 влияют на работу параметров и переменных
- Уметь пользоваться рекурсией для обхода «ограничений» управляющих элементов XSLT

Элементы программной логики в XSLT

В XSLT существуют три элемента, используемых для организации программной логики: <xsl:if>, <xsl:choose> и <xsl:for-each>. Первые два очень похожи на команды if и case, знакомые по другим языкам программирования, но элемент for-each существенно отличается от структур for или do-while в других языках. В этом разделе мы рассмотрим все перечисленные элементы.

Элемент <xsl:if>

Элемент <xsl:if> выглядит так:

```
<xsl:if test="count(zone) > 2">
  <xsl:text>Applicable zones: </xsl:text>
  <xsl:apply-templates select="zone"/>
</xsl:if>
```

Элемент <xsl:if> реализует функциональность конструкции if..., что в общем-то вполне естественно. Он имеет всего один атрибут – test. Если в результате обработки значения test будет получен результат true, то все элементы внутри <xsl:if> обрабатываются. Если вычисление test дает значение false, содержимое элемента <xsl:if> игнорируется. (Если вас интересует реализация if-then-else, обращайтесь к разделу «Элемент <xsl:choose>» далее в этой главе.)

Обратите внимание на использование символа > в значении атрибута test. Если потребуется использовать оператор «меньше» (<), его придется заменить сущностью <. То же относится к оператору «меньше или равно» (<=).

Преобразование к логическим значениям

Элемент <xsl:if> относительно прост, но при знакомстве с ним мы впервые сталкиваемся с *логическими значениями*. Они все равно встретятся нам позднее, поэтому можно обсудить их прямо сейчас. Такие атрибуты, как атрибут test элемента <xsl:if>, преобразуют свое значение в логическое (булево) значение. Если логическое значение равно true (логическая истина), элемент <xsl:if> обрабатывается. (Элемент <xsl:when>, который будет рассматриваться позднее в разделе «Элемент <xsl:choose>», также обладает атрибутом test.)

[1.0] Краткая сводка правил преобразования разных типов данных в логические значения:

число

Если число является положительным или отрицательным нулем, оно интерпретируется как false. Если число является NaN (нечисло; например, при попытке использовать строку "blue" как число будет получен результат NaN), оно интерпретируется как false. Все остальные значения интерпретируются как true.

набор узлов

Пустой набор узлов интерпретируется как `false`, непустой – как `true`.

строка

Строка нулевой длины интерпретируется как `false`; строка с ненулевой длиной интерпретируется как `true`.

Эти правила определяются в разделе 4.3 спецификации XPath 1.0.

[2.0] В XSLT 2.0 ситуация усложняется. Возвращаемое значение, как и следовало ожидать, относится к типу `xs:boolean`. А вот как разные типы данных преобразуются в значения `xs:boolean`:

- Синглет любого числового типа интерпретируется как `false`, если его значение равно нулю или `NaN` («нечисло»); все остальные значения интерпретируются как `true`.
- Синглеты типов `xs:string`, `xs:anyURI`, `xs:untypedAtomic` или любого типа, производного от них, интерпретируются как `true`, если длина аргумента больше нуля.
- Синглетное значение типа `xs:boolean` (или типа, производного от `xs:boolean`) используется без дополнительных преобразований.
- Последовательность, первый объект которой является узлом, интерпретируется как `true`.
- Пустая последовательность интерпретируется как `false`.
- При попытке преобразования любого типа данных в `xs:boolean` процессор XSLT выдает ошибку. Например, значение `xs:date` не может быть преобразовано в `true` или `false`.

Примеры

Рассмотрим некоторые примеры использования логических значений для проверки атрибута `test`:

```
<xsl:if test="count(zone) &gt;= 2">
```

Выражение является логическим, потому что в нем используется оператор «больше чем». Если функция `count()` возвращает значение большее либо равное 2, атрибут `test` равен `true`. В противном случае атрибут `test` равен `false`.

```
<xsl:if test="$x">
```

Переменная `$x` преобразуется в логическое значение по только что изложенным правилам. Конечно, результат зависит от значения переменной `$x` и способа ее преобразования в логическое значение.

```
<xsl:if test="true()">
```

Логическая функция `true()` всегда возвращает логическое значение `true`. Следовательно, атрибут `test` всегда равен `true`.

`<xsl:if test="true">`

Пример с подвохом: атрибут `test` интерпретируется как `true` только при наличии хотя бы одного элемента `<true>`, являющегося дочерним элементом контекстного узла. Процессор XSLT интерпретирует значение `true` как выражение XPath, определяющее все элементы `<true>` в текущем контексте. Строки `true` и `false` не имеют специальной интерпретации в XSLT.

`<xsl:if test="'true'">`

Этот атрибут `test` всегда имеет истинное значение. Обратите внимание: апострофы в кавычках указывают на то, что символы определяют литеральную строку, а не имя элемента. Этот атрибут `test` всегда истинен – но только потому, что длина строки больше нуля, а не потому, что он содержит слово «true».

`<xsl:if test="'false'">`

Еще один «пример с подвохом»: этот атрибут `test` тоже всегда истинен. Как и в предыдущем примере, апострофы в кавычках обозначают литеральную строку. Так как длина строки больше нуля, атрибут `test` всегда истинен. Значение непустой строки, каким бы странным оно ни было, роли не играет.

`<xsl:if test="not(3)">`

Этот атрибут `test` всегда равен `false`. Литерал `3` всегда интерпретируется как `true`, поэтому его отрицание равно `false`. С другой стороны, выражения `not(0)` и `not(-0)` всегда истинны.

`<xsl:if test="false( )">`

Атрибут `test` всегда равен `false`. Логическая функция `false()` всегда возвращает логическое значение `false`.

`<xsl:if test="section/section">`

Выражение XPath `section/section` возвращает набор узлов. Если текущий контекст содержит один или несколько элементов `<section>`, которые в свою очередь содержат элемент `<section>`, атрибут `test` равен `true`. Если в текущем контексте такие элементы отсутствуют, атрибут `test` равен `false`.

Элемент <xsl:choose>

Элемент `<xsl:choose>` логически эквивалентен конструкции `if-then-else`, хотя он также немного напоминает конструкции `case` и `switch` в других языках программирования. Элемент `<xsl:choose>` содержит как минимум один элемент `<xsl:when>` (логически эквивалентный элементу `<xsl:if>`) с необязательным элементом `<xsl:otherwise>` (логически эквивалентным `else` в других языках программирования). Атрибуты `test` всех элементов `<xsl:when>` последовательно вычисляются до тех пор, пока процессор XSLT не обнаружит элемент, равный `true`. Когда это произойдет, процессор обработает содержимое найденного эле-

мента <xsl:when>. (В отличие от конструкций case и switch, каждый элемент <xsl:when> является отдельной проверкой.) Если ни для одного элемента <xsl:when> при вычислении условия не был получен результат true, обрабатывается содержимое элемента <xsl:otherwise> (если он имеется).

Пример использования <xsl:choose>

В следующем примере элемент <xsl:choose> задает цвет фона строки таблицы. Если в элементе <table-row> закодирован атрибут bgcolor, его значение выбирается в качестве цвета фона; в противном случае функция position() и оператор mod последовательно переключаются между черным, зеленым, красным и синим цветом:

```
<xsl:template match="table-row">
  <tr>
    <xsl:attribute name="bgcolor">
     <xsl:choose>
        <xsl:when test="@bgcolor">
          <xsl:value-of select="@bgcolor"/>
        </xsl:when>
        <xsl:when test="position() mod 4 = 0">
          <xsl:text>black</xsl:text>
        </xsl:when>
        <xsl:when test="position() mod 4 = 1">
          <xsl:text>green</xsl:text>
        </xsl:when>
        <xsl:when test="position() mod 4 = 2">
          <xsl:text>red</xsl:text>
        </xsl:when>
        <xsl:otherwise>
          <xsl:text>blue</xsl:text>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:attribute>
    <xsl:apply-templates select="*"/>
  </tr>
</xsl:template>
```

В этом примере элемент <xsl:choose> генерирует значение атрибута bgcolor элемента <tr>. Первая проверка определяет, существует ли атрибут bgcolor элемента <table-row>; если он существует, его значение используется в качестве цвета фона, а <xsl:otherwise> и другие элементы <xsl:when> игнорируются. (Если атрибут bgcolor закодирован, выражение XPath @bgcolor возвращает набор узлов, состоящий из единственного узла атрибута.)

Следующие три элемента <xsl:when> проверяют позицию текущего элемента <table-row>. Оператор mod в данном случае обеспечивает наиболее эффективный способ переключения между несколькими фиксированными вариантами. Наконец, элемент <xsl:otherwise> задает синий цвет

по умолчанию. В случае `position() mod 4 = 3` будет использоваться синий цвет фона.

Также стоит обратить внимание на пару второстепенных деталей. В этом примере элемент `<xsl:otherwise>` можно заменить элементом `<xsl:when test="position() mod 4 = 3">`, в данном случае он будет логически эквивалентен. Если вам вдруг захочется поразвлечься с написанием запутанного кода, закодируйте второй элемент `<xsl:when>` в виде `<xsl:when test="not(position() mod 4)">`. (Помните, что логическое отрицание нуля соответствует `true`.)

Элемент <xsl:for-each>

Для обработки всех узлов, удовлетворяющих некоторому критерию, можно воспользоваться элементом `<xsl:for-each>`. Учтите, что он не является аналогом традиционных циклов `for`; нельзя приказать процессору XSLT выполнить вычисления вида:

```
for i = 1 to 10 do
```

Элемент `<xsl:for-each>` позволяет выбрать набор узлов, а затем выполнить некоторую операцию с каждым из выбранных узлов. Еще раз подчеркну, что этот цикл не является аналогом традиционных циклов `for`. Другой важный момент заключается в том, что текущий узел изменяется при каждой итерации элемента `<xsl:for-each>`. Мы рассмотрим несколько примеров, демонстрирующих эти особенности `<xsl:for-each>`.

Оператор XSLT 2.0 `to` (`select "1 to 10"`) позволяет реализовать нечто похожее на традиционный цикл `for`. При работе с XSLT лучше рассматривать `<xsl:for-each>` как итератор, а не как традиционный цикл `for`.

Пример <xsl:for-each>

Следующий пример выбирает все элементы `<section>` в элементе `<tutorial>`, а затем использует второй элемент `<xsl:for-each>` для выбора всех элементов `<panel>` в каждом элементе `<section>`:

```
<xsl:template match="tutorial">
  <xsl:for-each select="section">
    <h1>
      <xsl:text>Section </xsl:text>
      <xsl:value-of select="position()"/>
      <xsl:text>. </xsl:text>
      <xsl:value-of select="title"/>
    </h1>
    <ul>
      <xsl:for-each select="panel">
        <li>
          <xsl:value-of select="position()"/>
          <xsl:text>. </xsl:text>
          <xsl:value-of select="title"/>
```

```
        </li>
      </xsl:for-each>
    </ul>
  </xsl:for-each>
</xsl:template>
```

Для следующего документа XML:

```
<tutorial>
  <section>
    <title>Gene Splicing for Young People</title>
    <panel>
      <title>Introduction</title>
      <!-- ... -->
    </panel>
    <panel>
      <title>Discovering the secrets of life and creation</title>
      <!-- ... -->
    </panel>
    <panel>
      <title>"I created him for good, but he's turned out evil!"</title>
      <!-- ... -->
    </panel>
    <panel>
      <title>When angry mobs storm your castle</title>
      <!-- ... -->
    </panel>
  </section>
</tutorial>
```

представленный шаблон выдает следующий результат:

```
<h1>Section 1. Gene Splicing for Young People</h1>
<ul>
  <li>1. Introduction</li>
  <li>2. Discovering the secrets of life and creation</li>
  <li>3. "I created him for good, but he's turned out evil!"</li>
  <li>4. When angry mobs storm your castle</li>
</ul>
```

Каждый атрибут `select` обрабатывается в контексте текущего узла. По мере того как процессор XSLT перебирает все элементы `<section>` и `<panel>`, каждый из них последовательно становится текущим узлом. Итерация позволила нам сгенерировать сводку содержимого при помощи очень простого шаблона.

Вызов шаблонов по имени

Во всех предыдущих примерах для активизации других шаблонов применялся элемент XSLT `<xsl:apply-templates>`. Происходящее можно рассматривать как простейшую форму *полиморфизма*: одна и та же команда вызывается многократно, а процессор XSLT в зависимости от те-

кущего узла в наборе определяет, какой элемент <xsl:template> следует активизировать. Обычно желательно именно такое поведение, но в некоторых ситуациях требуется активизировать конкретный шаблон. В XSLT эта задача решается при помощи элемента <xsl:call-template>.

Как это делается

Для активизации шаблона по имени должны выполняться два условия:

- Шаблон должен обладать именем (name)
- Именованный шаблон активизируется элементом <xsl:call-template>

Конкретный пример: допустим, имеется шаблон createMasthead для создания заголовка веб-страницы. Каждый раз, когда мы создаем страницу HTML для своего сайта, таблица стилей должна строить заголовок вызовом шаблона createMasthead. Вот как выглядит таблица стилей:

```
<xsl:template name="createMasthead">
  <!-- Здесь генерируется заголовок -->
</xsl:template>
...
<xsl:template match="/">
  <html>
    <head>
      <title><xsl:value-of select="title"/></title>
    </head>
    <body>
      <xsl:call-template name="createMasthead"/>
...
```

Именованные шаблоны чрезвычайно удобны для определения стандартной разметки. Допустим, таблица стилей используется для создания веб-страниц с определенным оформлением. Вы можете написать именованные шаблоны для создания заголовка и завершителя, областей навигации и других элементов, определяющих внешний вид страницы. Каждый раз, когда потребуется создать веб-страницу, просто используйте элемент <xsl:call-template> для вызова этих шаблонов и создания нужных компонентов оформления.

Еще лучше разместить именованные шаблоны в отдельной таблице стилей и импортировать ее (элементом <xsl:import> или <xsl:include>) – это позволит создать набор таблиц стилей, генерирующих нужное оформление. Если вы решите изменить дизайн своего сайта, перепишите таблицы стилей, определяющие общие графические элементы и общую структуру. Измените таблицы стилей, сгенерируйте заново сайт – *voila*! Ваш сайт мгновенно преобразился.

Шаблоны и атрибут mode

Элемент XSLT <xsl:template> содержит атрибут mode, управляющий многократной обработкой одного набора узлов. Допустим, элементы <h1> должны обрабатываться одним способом при генерировании сводки содержимого и другим при обработке документа в целом. Атрибут mode позволяет определить разные шаблоны для разных целей:

```
<xsl:template match="h1" mode="build-toc">
  <!-- Шаблон для обработки элемента <h1> при построении сводки содержимого -
->
</xsl:template>

<xsl:template match="h1" mode="process-text">
  <!-- Шаблон для обработки элементов <h1> в остальных   -->
  <!-- частях документа                                  -->
</xsl:template>
```

Далее шаблоны применяются с указанием соответствующего значения атрибута mode:

```
<xsl:template match="/">
  <html>
    <body>
      <h1>Table of Contents</h1>
      <ul>
        <xsl:apply-templates select="h1" mode="build-toc"/>
      </ul>
      <xsl:apply-templates select="*" mode="process-text"/>
    </body>
  </html>
</xsl:template>
```

Такой стиль программирования заметно упрощает сопровождение кода; если сводка содержимого сгенерирована неверно, очевидно, отладку следует начать с шаблонов mode="build-toc". (Атрибут mode более подробно описан в разделе «Новые значения атрибута mode» далее в этой главе.)

Параметры

Элементы XSLT <xsl:param> и <xsl:with-param> дают возможность передавать параметры при вызове шаблона. Параметры могут передаваться как с элементом <call-template>, так и с элементом <apply-templates>; механизм передачи параметров более подробно описан в этом разделе.

Определение параметра в шаблоне

Для определения параметра в шаблоне используется элемент <xsl:param>. Пример шаблона с определением двух параметров:

```
<xsl:template name="calcuateArea">
  <xsl:param name="width"/>
  <xsl:param name="height"/>
  <xsl:value-of select="$width * $height"/>
</xsl:template>
```

На концептуальном уровне очень напоминает программу на традиционном языке программирования, не так ли? Шаблон определяет два параметра, `width` и `height`, и выводит их произведение.

При желании для параметра можно определить значение по умолчанию. Существуют два способа определения значений по умолчанию; более простой основан на использовании атрибута `select` элемента `<xsl:param>`:

```
<template name="addTableCell">
  <xsl:param name="bgColor" select="'blue'"/>
  <xsl:param name="width" select="150"/>
  <xsl:param name="content"/>
  <td width="{$width}" bgcolor="{$bgColor}">
    <xsl:apply-templates select="$content"/>
  </td>
</template>
```

В этом примере параметрам `bgColor` и `width` назначаются значения по умолчанию `'blue'` и `150` соответственно. Если при вызове шаблона значения этих параметров не указаны, будут использованы значения по умолчанию. Также обратите внимание на то, что значения атрибутов `width` и `bgColor` элемента HTML `<td>` генерируются с использованием шаблонов значений атрибутов (значения в фигурных скобках). За дополнительной информацией обращайтесь к разделу «Шаблоны значений атрибутов» главы 3.

И последнее, на что следует указать в этом примере: параметр `content` не имеет значения по умолчанию; предполагается, что `content` содержит узлы, которые будут обработаны и помещены в ячейку таблицы. Если значение `content` является пустой строкой, при вызове `<xsl:apply-templates select="$content"/>` будет получена ошибка. Для надежности можно было бы добавить элемент `<xsl:if>`, который создает пустую ячейку таблицы, если `content` является пустой строкой. Если вам захочется сделать этот код более надежным, пусть это будет упражнением для самостоятельной работы...

Обратите внимание: в приведенном примере значение `blue` заключено в апострофы, а значение `150` – нет. В случае отсутствия апострофов вокруг `blue` процессор XSLT решит, что мы хотим выбрать все элементы `<blue>` в текущем контексте... а это совсем не то, что требуется. Однако процессор XSLT понимает, что `150` не может быть именем элемента XML (в спецификации XML 1.0 сказано, что имена элементов не могут начинаться с цифр), поэтому заключать числовые значения в апострофы необязательно.

Постарайтесь помнить об этом при использовании параметров. Впрочем, рано или поздно вы все равно об этом забудете и станете сходить с ума, пытаясь разобраться в странном поведении процессора XSLT.

Второй способ определения значения по умолчанию для параметра основан на включении контента в элемент <xsl:param>:

```
<template name="addTableCell">
  <xsl:param name="bgColor">
    <xsl:text>blue</xsl:text>
  </xsl:param>
  <xsl:param name="width">
    <xsl:value-of select="7+8"/><xsl:text>0</xsl:text>
  </xsl:param>
  <xsl:param name="content"/>
  <td width="{$width}" bgcolor="{$bgColor}">
    <xsl:apply-templates select="$content"/>
  </td>
</template>
```

В этом примере элементы <xsl:text> и <xsl:value-of> используются для определения значений по умолчанию. Просто для интереса мы определяем значение width как результат конкатенации числового выражения 7+8 со строкой 0. Результат числового выражения 15 преобразуется в строку, а затем эта строка объединяется со строкой 0. В результате будет получена строка 150, которая затем преобразуется в число.

Передача параметров

При вызове шаблона по имени (что аналогично вызову функции) обычно требуется задать значения передаваемых параметров. Для этой цели используется элемент <xsl:with-param>. Допустим, мы хотим вызвать шаблон draw-box и передать ему параметры startX, startY, endX и endY. Это делается так:

```
<xsl:call-template name="draw-box">
  <xsl:with-param name="startX" select="50"/>
  <xsl:with-param name="startY" select="50"/>
  <xsl:with-param name="endX" select="97"/>
  <xsl:with-param name="endY" select="144"/>
</xsl:call-template>
```

В этом примере шаблон draw-box вызывается с четырьмя параметрами. Обратите внимание: ранее тег <xsl:call-template> всегда был пустым, а теперь в нем появился вложенный контент – определения параметров. (При желании то же самое можно сделать и с элементом <xsl:apply-templates>.)

Если вы собираетесь передавать параметры шаблону, его необходимо заранее настроить так, чтобы он был готов принять передаваемые параметры. Для этого в шаблон включается элемент <xsl:param>. Несколько примеров:

```
<xsl:template name="draw-box">
  <xsl:param name="startX"/>
  <xsl:param name="startY" select="'0'"/>
  <xsl:param name="endX">
    10
  </xsl:param>
  <xsl:param name="endY">
    10
  </xsl:param>
  ...
</xsl:template>
```

Несколько замечаний по поводу элемента <xsl:param>:

- Если в шаблоне определяются элементы <xsl:param>, они должны быть первыми элементами в шаблоне.
- Элемент <xsl:param> позволяет задать значение по умолчанию для параметра. Если вызывающий шаблон не указывает значение при вызове, вместо него используется значение по умолчанию. Три последних элемента <xsl:param> в нашем примере определяют значения по умолчанию.
- Элемент <xsl:param> использует ту же модель контента, что и <xsl:variable>. Без контента и без атрибута select значение параметра по умолчанию представляет собой пустую строку (""). С атрибутом select значение параметра по умолчанию соответствует значению атрибута select. Если элемент <xsl:param> содержит контент, то значение параметра по умолчанию определяется контентом элемента <xsl:param>.

Глобальные параметры

XSLT позволяет определять параметры, область видимости которых распространяется на всю таблицу стилей. Для таких параметров можно определять значения по умолчанию, а также получать их значения из внешних источников. Но прежде чем обсуждать механизм передачи значений *глобальных параметров*, разберемся, как создаются такие параметры. Все параметры, являющиеся элементами верхнего уровня (т. е. любые элементы <xsl:param>, родителем которых является элемент <xsl:stylesheet>), считаются глобальными. Пример:

```
<?xml version="1.0"?>
<!-- params.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:param name="startX"/>
  <xsl:param name="baseColor"/>

  <xsl:template match="/">
```

```
    <xsl:text>&#xA;Global parameters example&#xA;&#xA;</xsl:text>

    <xsl:text>The value of startX is: </xsl:text>
    <xsl:value-of select="$startX"/>
    <xsl:text>&#xA;The value of baseColor is: </xsl:text>
    <xsl:value-of select="$baseColor"/>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Механизм передачи значений глобальных параметров зависит от используемого процессора XSLT. Мы рассмотрим примеры для всех основных кандидатов. Предположим, вы хотите передать числовое значение `50` в качестве значения `startX`, а строковое значение `magenta` должно использоваться по умолчанию для параметра `baseColor`. В следующем списке представлены команды передачи параметров для всех основных процессоров XSLT.

Xalan

В Xalan глобальные параметры могут определяться в командной строке:

```
java org.apache.xalan.xslt.Process -in blank.xml -xsl params.xsl
  -param startX 50 -param baseColor magenta
```

(Команда должна вводиться в одной строке.)

Saxon

Saxon поддерживает внешние параметры со следующим синтаксисом:

```
java net.sf.saxon.Transform blank.xml params.xsl startX=50
  baseColor=magenta
```

Microsoft XSLT Processor

Передача внешних параметров Microsoft XSLT Processor:

```
msxsl blank.xml params.xsl startX=50 baseColor=magenta
```

Altova

Если вы используете ядро XSLT Altova, используйте внешние параметры следующего вида:

```
altovaxml /xslt1 params.xsl /in blank.xml /param startX=50 /param
baseColor='magenta'
```

Обратите внимание на апострофы, в которые заключено текстовое значение `magenta`.

При использовании этой таблицы стилей с любым документом XML и любым из перечисленных процессоров XSLT будет получен следующий результат:

```
Global parameters example

The value of startX is: 50
The value of baseColor is: magenta
```

Глобальные параметры в программах Java

Если ваш процессор XSLT поддерживает TrAX (Transformation API for XML), глобальные параметры можно задать на уровне программного кода. Пример использования поддержки TrAX:

```
import java.io.File;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;

public class GlobalParameters
{
  public static void parseAndProcess(String sourceID,
                                     String xslID,
                                     String outputID)
  {
    try
    {
      TransformerFactory tfactory = TransformerFactory.newInstance();

      Transformer transformer
        = tfactory.newTransformer(new StreamSource(xslID));

     // Задание глобальных параметров методом setParameter
      transformer.setParameter("startX", new Integer(50));
      transformer.setParameter("baseColor", "magenta");

      transformer.transform(new StreamSource(sourceID),
                            new StreamResult(outputID));
    }
    catch (TransformerConfigurationException tce)
    {
      System.err.println("Exception: " + tce);
    }
    catch (TransformerException te)
    {
      System.err.println("Exception: " + te);
    }
  }

  public static void main(String argv[])
    throws java.io.IOException,
           org.xml.sax.SAXException
  {
    GlobalParameters gp = new GlobalParameters();
    gp.parseAndProcess(argv[0], argv[1], argv[2]);
```

```
    }
}
```

Обратите внимание: метод `setParameter` **задает глобальные параметры объекта** `Transformer` **перед вызовом метода** `transform`**. Преобразование генерирует следующий результат в файле** `output.text`**:**

```
Global parameters example

The value of startX is: 50
The value of baseColor is: magenta
```

Глобальные параметры в .NET

Инфраструктура .NET предоставляет объект `XsltArgumentList` **для задания глобальных параметров таблиц стилей. Как и в аналогичном примере Java, программный код весьма прямолинеен:**

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Xml;
using System.Xml.Xsl;

namespace com.oreilly.xslt
{
  class XsltGlobalParameters
  {
    static void Main(string[] args)
    {
      // Создание объекта таблицы стилей и объекта
      // XMLWriter для записи вывода в файл
      XslCompiledTransform stylesheet = new XslCompiledTransform();
      XmlTextWriter xWriter =
        new XmlTextWriter(args[2], Encoding.UTF8);

      // Используем объект XsltSettings, допускающий выполнение сценариев
      // (это необходимо для расширений), затем загружаем таблицу стилей.
      XsltSettings settings = new XsltSettings(true, true);
      stylesheet.Load(args[1], settings, new XmlUrlResolver());

      // Передаем глобальные параметры таблице стилей
      // с объектом XsltArgumentList.
      XsltArgumentList argList = new XsltArgumentList();
      argList.AddParam("startX", "", 50);
      argList.AddParam("baseColor", "", "magenta");

      // После завершения подготовки вызывается метод Transform(),
      // который выполняет всю основную работу...
      stylesheet.Transform(args[0], argList, xWriter);
    }
  }
}
```

Мы создаем объект `XslCompiledTransform`, а затем используем объект `XsltArgumentList` для задания параметров таблицы стилей. После завершения всей предварительной настройки вызывается метод `Transform()` класса `XslCompiledTransform`. Программа выдает тот же результат, который мы уже видели в программном коде Java:

```
Global parameters example

The value of startX is: 50
The value of baseColor is: magenta
```

[2.0] Важные изменения в XSLT 2.0

В XSLT 2.0 в области использования параметров и атрибута `mode` были внесены некоторые изменения. Краткая сводка основных нововведений:

- Атрибут `mode` поддерживает три новых значения: `#all`, `#current` и `#default`.
- Если шаблону передается параметр, который не определен в шаблоне, процессор XSLT 2.0 выдает сообщение об ошибке и прекращает работу. В XSLT 1.0 неопределенные параметры просто игнорируются.
- В XSLT 2.0 добавлен атрибут `required="yes"`, указывающий, что значение параметра обязательно должно передаваться при вызове.
- Появилась возможность задания типа данных и/или структуры параметра. Например, если параметр должен относиться к типу `xs:date` или представлять собой последовательность, содержащую, как минимум, один элемент, вы можете задать это условие.
- В XSLT 2.0 появилась новая разновидность параметров – так называемые *туннельные параметры*. Туннельные параметры помогают избежать неудобных программных «выкрутасов», которые часто были необходимы в XSLT 1.0.

Новые значения атрибута mode

В XSLT 2.0 атрибут `mode` может принимать три новых значения:

`#all`

В элементе `<xsl:template>` может использоваться новое значение `mode="#all"`. Оно указывает, что данный шаблон соответствует любому значению `mode`. Впрочем, для текущего режима `"toc"` будет активизирован шаблон с атрибутом `mode="toc"` (вместо шаблона с атрибутом `mode="#all"`).

`#current`

В элементе `<xsl:apply-templates>` значение `mode="#current"` может использоваться для вызова других шаблонов с использованием текущего значения `mode`. Фактически при этом текущий режим неявно передается в дополнительном параметре.

`#default`

Элементы `<xsl:apply-templates>` и `<xsl:template>` могут использовать значение `mode="#default"` для режима по умолчанию.

Другое отличие XSLT 2.0: значение атрибута `mode` может задаваться в виде списка имен режимов, разделенных пробелами. В XSLT 1.0 может задаваться только одно значение `mode`.

Полный пример использования новых возможностей приведен в описании элемента `<xsl:apply-templates>` в приложении А.

Неопределенные параметры запрещены

В XSLT 1.0 шаблону можно передать сколько угодно параметров. Если передать два параметра шаблону, определяющему всего один параметр, лишний параметр просто игнорируется. Процессор XSLT даже не выдаст предупреждения или сообщения об ошибке. Пример:

```
<?xml version="1.0"?>
<!-- parameters-1_0.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:call-template name="test">
      <xsl:with-param name="param1" select="'57'"/>
      <xsl:with-param name="param2" select="'93'"/>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="test">
    <xsl:param name="param1"/>
    <xsl:text>Value of $param1: </xsl:text>
    <xsl:value-of select="$param1"/>
  </xsl:template>

</xsl:stylesheet>
```

В этом примере шаблон корня документа вызывает шаблон по имени и передает ему два параметра. Параметр `param2` не определяется в вызываемом шаблоне. С процессором XSLT 1.0 ошибки при этом *не* возникает. Вот как выглядит результат, полученный от Xalan-J 2.7.0, который является процессором XSLT 1.0:

```
java org.apache.xalan.xslt.Process -xsl parameters-1_0.xsl
Value of $param1: 57
```

При использовании той же таблицы стилей с Saxon 9.0.0.3, который является процессором XSLT 2.0, результат будет таким:

```
java net.sf.saxon.Transform blank.xml parameters-1_0.xsl
Warning: Running an XSLT 1.0 stylesheet with an XSLT 2.0 processor
Value of $param1: 57
```

Saxon предупреждает о том, что он выполняет таблицу стилей XSLT 1.0, но игнорирует лишний параметр, как это и должно быть при обработке таблицы стилей в режиме XSLT 1.0. Но если привести атрибут `version` в теге `<xsl:stylesheet>` к следующему виду:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">
```

Saxon обработает таблицу стилей по правилам XSLT 2.0:

```
java net.sf.saxon.Transform blank.xml parameters-2_0.xsl
Error at xsl:call-template on line 9 of file:/C:/parameters-2_0.xsl:
  XTSE0680: Parameter param2 is not declared in the called template
Failed to compile stylesheet. 1 error detected.
```

Процессор XSLT 2.0 прекращает преобразование сразу же после обнаружения неопределенного параметра.

Наконец, при обработке таблицы стилей XSLT 2.0 в Xalan атрибут `version="2.0"` полностью игнорируется, а таблица стилей обрабатывается по правилам XSLT 1.0:

```
java org.apache.xalan.xslt.Process -xsl parameters-2_0.xsl
Value of $param1: 57
```

Возможно, вы заметили, что при обработке таблицы стилей в Xalan задается только таблица стилей. С другой стороны, Saxon требует, чтобы задавался как файл XML, так и таблица стилей XSLT. Содержимое файла *blank.xml* состоит из единственного пустого элемента `<blank/>`.

Другое возможное решение – присвоить шаблону имя (`<xsl:template match="/" name="main">`) и активизировать его при помощи параметра Saxon `-it`:

```
java net.sf.saxon.Transform -it main parameters_1.0.xsl
Warning: Running an XSLT 1.0 stylesheet with an XSLT 2.0 processor
Value of $param1: 57
```

Параметр `-it` задает именованный шаблон, с которого должно начаться преобразование.

Обязательные параметры

XSLT 2.0 дополняет элемент `<xsl:param>` атрибутом `required`. Как нетрудно предположить, допустимые значения этого атрибута – `yes` и `no`. Для обязательных параметров (`required="yes"`) элемент `<xsl:param>` *не может* содержать атрибут `select`.

Пример обязательного параметра:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- required_parameters.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>
```

```
  <xsl:template match="/">
    <xsl:call-template name="date-formatter">
      <xsl:with-param name="date" select="current-date()"/>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="date-formatter">
    <xsl:param name="date" required="yes"/>
    <xsl:value-of select="format-date($date, '[M01]/[D01]/[Y0001]')"/>
  </xsl:template>

</xsl:stylesheet>
```

В этой таблице стилей параметр `date` является обязательным; если вы попытаетесь вызвать шаблон без передачи обязательного параметра, процессор XSLT откажется обрабатывать таблицу. Здесь также было бы уместно использовать поддержку типизации данных XSLT 2.0 (подробности позже); процессор XSLT заставляет передать обязательный параметр, но не проверяет его тип данных. Если параметр, переданный функции `format-date()`, не относится к типу `xs:date`, процессор XSLT выдает ошибку. (Таблица стилей генерирует параметр при помощи функции `current-date()`, поэтому в нашем примере тип данных заведомо правилен.)

Последнее ограничение `<xsl:param>` заключается в том, что к одному и тому же параметру не могут применяться атрибуты `required` и `select`. Вспомните, что атрибут `select` определяет значение по умолчанию на тот случай, если параметр не передается шаблону. Атрибут `required`, напротив, сообщает, что параметр *должен* передаваться шаблону.

Поддержка типизации данных

Элементы `<xsl:param>`, `<xsl:with-param>` и `<xsl:variable>` поддерживают необязательный атрибут `as`, определяющий тип данных и/или структуру параметра или переменной. В следующем примере параметр должен относиться к типу `xs:date`:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datatype_parameters.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:call-template name="date-formatter">
      <xsl:with-param name="date" select="current-date()"/>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="date-formatter">
    <xsl:param name="date" as="xs:date" required="yes"/>
```

```
    <xsl:value-of select="format-date($date, '[M01]/[D01]/[Y0001]')"/>
  </xsl:template>

</xsl:stylesheet>
```

Обычно проверяется тип данных параметра, но мы также можем определить его структуру. Атрибут `as="element()+"` указывает, что параметр представляет собой последовательность из одного или нескольких узлов элементов, тогда как атрибут `as="element(*, xs:date)+"` сообщает, что параметр представляет собой последовательность из одного или нескольких узлов элементов, каждый из которых относится к типу `xs:date`.

В этой таблице стилей вызывается именованный шаблон. Элемент `<xsl:param>` в именованном шаблоне сообщает процессору XSLT, что параметр является обязательным и что его типом данных должен быть тип XSL Schema `date`. При обработке этой таблицы стилей в процессоре XSLT 2.0 будет получен следующий результат:

```
java net.sf.saxon.Transform blank.xml datatype_parameters.xsl
03/02/2008
```

Ядро таблиц стилей XSLT 2.0 получает значение параметра и форматирует его в соответствии с форматной строкой, указанной при вызове функции `format-date()`.

Если тип данных параметра не соответствует типу, требуемому шаблоном, происходит ошибка. Например, если изменить таблицу стилей так, что шаблон будет вызываться с недопустимыми данными:

```
<xsl:call-template name="date-formatter">
  <xsl:with-param name="date" select="'blue'"/>
</xsl:call-template>
```

произойдет ошибка:

```
Error at xsl:with-param on line 11 of file:/C:/datatype_parameters.xsl:
  XTTE0570: Required item type of value of variable $date is xs:date;
  supplied value has item type xs:string
Failed to compile stylesheet. 1 error detected.
```

Ядро XSLT даже не обработает таблицу стилей, потому что значение параметра (строка `blue`) явно не совпадает с требуемым типом данных. Если значение параметра генерируется динамически и сгенерированное значение не соответствует типу `xs:date`, происходит ошибка времени выполнения, а преобразование прерывается.

Туннельные параметры

Работа с параметрами в XSLT 1.0 имеет некоторые недостатки. Во-первых, вы можете игнорировать «сигнатуру» вызываемого шаблона. При вызове можно передавать нужное количество параметров, передавать вдвое больше параметров или не передавать их вовсе. Вероятно,

процессор XSLT 1.0 сделает то, что вы хотели, но если параметр задан неверно, найти причины возникших проблем будет нелегко. XSLT 2.0 требует, чтобы при вызове передавалось ровно столько параметров, сколько требуется; это делает программный код более логичным.

Вторая проблема возникает тогда, когда передаваемый параметр может быть использован другим шаблоном. Допустим, у нас имеется таблица стилей, генерирующая код HTML на базе DocBook. Документ DocBook содержит сотни элементов, поэтому мы ограничимся шаблонами для небольшого подмножества элементов DocBook. Мы возьмем документ DocBook и создадим два файла HTML. Каждый файл HTML содержит заголовки основных разделов (`sect1/title`) и все листинги из исходного документа (элементы DocBook `<programlisting>`). Преобразование выполняется дважды: в одном документе генерируется текст обычного размера, а во втором – более крупный текст.

Вот как выглядит таблица стилей *без* туннельных параметров:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- normal_parameters.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <xsl:result-document href="regular-type.html" method="html">
      <html>
        <xsl:apply-templates select="*|text()">
          <xsl:with-param name="code-font-size" select="'14'"/>
        </xsl:apply-templates>
      </html>
    </xsl:result-document>
    <xsl:result-document href="larger-type.html" method="html">
      <html>
        <xsl:apply-templates select="*|text()">
          <xsl:with-param name="code-font-size" select="'20'"/>
        </xsl:apply-templates>
      </html>
    </xsl:result-document>
  </xsl:template>

  <xsl:template match="chapter">
    <xsl:param name="code-font-size"/>
    <head>
      <title><xsl:value-of select="title"/></title>
    </head>
    <body>
      <xsl:apply-templates select="*[not(name() = 'title')]|text()">
        <xsl:with-param name="code-font-size" select="$code-font-size"/>
      </xsl:apply-templates>
    </body>
  </xsl:template>
```

```
  <xsl:template match="programlisting">
    <xsl:param name="code-font-size"/>
    <pre>
      <span>
        <xsl:attribute name="style">
          <xsl:text>font-family:monospace; font-size:</xsl:text>
          <xsl:value-of select="$code-font-size"/>
          <xsl:text>;</xsl:text>
        </xsl:attribute>
        <xsl:apply-templates select="*|text()">
          <xsl:with-param name="code-font-size" select="$code-font-size"/>>
        </xsl:apply-templates>
      </span>
    </pre>
  </xsl:template>

  <xsl:template match="sect1/title">
    <xsl:param name="code-font-size"/>
    <h1>
      <xsl:apply-templates select="*|text()">
        <xsl:with-param name="code-font-size" select="$code-font-size"/>
      </xsl:apply-templates>
    </h1>
  </xsl:template>

  <!-- В реальной таблице стилей здесь следуют десятки шаблонов... -->
  <xsl:template match="*">
    <xsl:param name="code-font-size"/>
    <xsl:apply-templates select="*">
      <xsl:with-param name="code-font-size" select="$code-font-size"/>
    </xsl:apply-templates>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей генерирует нужные результаты и создает документы `regular-type.html` **и** `larger-type.html`**. Фрагмент файла** `regular-type.html` **может выглядеть так:**

```
        <h1>Goals of This Chapter</h1>

        <h1>Branching Elements of XSLT</h1><pre>
<span style="font-family:monospace; font-size:14;">
&lt;xsl:if test="count(zone) &gt; 2"&gt;
  &lt;xsl:text&gt;Applicable zones: &lt;/xsl:text&gt;
  &lt;xsl:apply-templates select="zone"/&gt;
&lt;/xsl:if&gt;</span></pre><pre>
<span style="font-family:monospace; font-size:14;">
&lt;xsl:template match="table-row"&gt;
  &lt;tr&gt;
    &lt;xsl:attribute name="bgcolor"&gt;
     &lt;xsl:choose&gt;
```

Файл larger-type.html идентичен за одним исключением: атрибут style содержит font-size:20; вместо font-size:14;. Однако таблица стилей получается громоздкой, а ее сопровождение потребует больших усилий, чем нам хотелось бы. Очень многие шаблоны имеют следующую структуру:

```
<xsl:template match="whatever">
  <xsl:param name="code-font-size"/>

  <!-- Что-то сделать с текущим элементом, -->
  <!-- затем обработать его потомков       -->

  <xsl:apply-templates select="*|text()">
    <!-- Передать параметр code-font-size parameter -->
    <!-- на случай, если он понадобится позднее.    -->
    <xsl:with-param name="code-font-size" select="$code-font-size"/>
  </xsl:apply-templates
</xsl:template>
```

Проблема в том, что при каждом использовании <xsl:apply-templates> нам приходится передавать переменную code-font-size — просто на случай, если она понадобится шаблону, расположенному где-то дальше в цепочке. Любой из элементов, для которых мы написали шаблоны, может иметь потомка <programlisting>, так что выбора у нас нет. Ситуация усугубляется тем, что при каждом добавлении нового шаблона в таблицу стилей приходится добавлять ту же разметку <xsl:param> и <xsl:with-param>. Если вдруг появятся еще три параметра, которые придется передавать подобным способом, таблица стилей окончательно запутается, а любое изменение в ней приведет к возникновению ошибок (если вы забудете повторить внесенные изменения во всех шаблонах, которые от них зависят).

В подобных ситуациях на помощь приходят туннельные параметры.

Туннельные параметры напоминают *переменные с динамической областью видимости* в функциональных языках программирования, таких как Haskell и Scheme. В этих языках переменная может входить в область видимости и выходить из нее, когда одна функция вызывает другую. В XSLT 2.0 туннельный параметр передается каждому шаблону, вызываемому явно или косвенно. Если один шаблон вызывает другой шаблон в ходе своего выполнения, то любой шаблон может использовать туннельный параметр, ссылаясь на него соответствующим образом (как на туннельный параметр). Вот как выглядит таблица стилей с туннельными параметрами:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- tunnel_parameters.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
```

```
    <xsl:result-document href="regular-type.html" method="html">
      <html>
        <xsl:apply-templates select="*|text()">
          <xsl:with-param name="code-font-size" select="'14'"
            tunnel="yes"/>
        </xsl:apply-templates>
      </html>
    </xsl:result-document>
    <xsl:result-document href="larger-type.html" method="html">
      <html>
        <xsl:apply-templates select="*|text()">
          <xsl:with-param name="code-font-size" select="'20'"
            tunnel="yes"/>
        </xsl:apply-templates>
      </html>
    </xsl:result-document>
  </xsl:template>

  <xsl:template match="chapter">
    <head>
      <title><xsl:value-of select="title"/></title>
    </head>
    <body>
      <xsl:apply-templates select="*[not(name() = 'title')]|text()"/>
    </body>
  </xsl:template>

  <xsl:template match="programlisting">
    <xsl:param name="code-font-size" tunnel="yes"/>
    <pre>
      <span>
        <xsl:attribute name="style">
          <xsl:text>font-family:monospace; font-size:</xsl:text>
          <xsl:value-of select="$code-font-size"/>
          <xsl:text>;</xsl:text>
        </xsl:attribute>
        <xsl:apply-templates select="*|text()"/>
      </span>
    </pre>
  </xsl:template>

  <xsl:template match="sect1/title">
    <h1>
      <xsl:apply-templates select="*|text()"/>
    </h1>
  </xsl:template>

  <!-- В реальной таблице стилей здесь следуют десятки шаблонов... -->
  <xsl:template match="*">
    <xsl:apply-templates select="*"/>
  </xsl:template>

</xsl:stylesheet>
```

Обратите внимание, насколько туннельные параметры упростили код. В корневом элементе туннельный параметр передается при запросе к процессору XSLT на преобразование всех элементов-потомков. Каждый раз, когда последующий шаблон вызывает другой шаблон посредством <apply-templates> или <call-template>, туннельные параметры незаметно передаются вызываемому шаблону. В таблице стилей этот параметр используется в единственном месте, где он действительно необходим: в шаблоне элемента programlisting.

Пара замечаний по поводу синтаксиса: прежде всего, у туннельных параметров элемент <xsl:with-param>, в котором параметр объявляется, должен содержать атрибут tunnel="yes". Во-вторых, шаблон, который хочет использовать туннельный параметр, должен содержать атрибут tunnel="yes" в элементе <xsl:param>, определяющем параметр. Если определение параметра не содержит атрибута tunnel="yes", процессор XSLT считает, что параметр является новой переменной, локальной для данного шаблона.

Казалось бы, проблему можно решить и при помощи глобального параметра. И это действительно так, однако такое решение имеет пару недостатков. Прежде всего, количество глобальных параметров желательно свести к минимуму. Добавление глобального параметра, который может использоваться где угодно, не соответствует канонам хорошего стиля программирования.

Вторая проблема заключается в том, что значение глобального параметра нельзя изменять. В нашем примере придется создавать две глобальные переменные: по одной для каждого значения, используемого в таблице стилей. Туннельный параметр позволяет задавать значение code-font-size каждый раз, когда потребуется обработать исходный документ. Туннельные параметры упрощают написание, отладку и сопровождение таблиц стилей.

Переменные

Если для управления ходом обработки таблиц стилей используются логические элементы, вероятно, в какой-то момент мы захотим сохранить промежуточные результаты. Другими словами, мы должны использовать переменные. XSLT предоставляет элемент <xsl:variable>, который позволяет сохранить значение и связать его с именем.

Существуют три варианта использования элемента <xsl:variable>. В простейшей форме элемент создает новую переменную, значением которой является пустая строка (""):

```
<xsl:variable name="x"/>
```

Элемент создает новую переменную x, значением которой является пустая строка. (Аплодисменты лучше отложить до конца раздела.)

Во втором варианте создания переменной в элемент <xsl:variable> включается атрибут select:

```
<xsl:variable name="favouriteColour" select="'blue'"/>
```

В этом примере переменная инициализируется строкой «blue». Обратите внимание на апострофы, в которые заключено значение переменной. Они гарантируют, что в качестве значения переменной будет использовано литеральное значение blue. Если убрать апострофы, процессор XSLT посчитает, что переменная инициализируется набором узлов (или последовательностью) всех элементов <blue> в контекстном узле, а это определенно не то, что нам нужно.

Апострофы, в которые заключаются числовые значения, важны. Значение 35 представляет числовое значение (это число в XSLT 1.0 и xs:integer в XSLT 2.0), а значение '35' представляет *строку* 35. На первый взгляд это различие несущественно, но оно оказывает серьезное влияние на работу таблицы стилей, особенно в XSLT 2.0.

В третьей форме в элементе <xsl:variable> используется вложенный контент. Короткий пример:

```
<xsl:variable name="y">
  <xsl:choose>
    <xsl:when test="$x &gt; 7">
      <xsl:text>13</xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>15</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable>
```

В этом (чуть более сложном) примере содержимое переменной y зависит от атрибута test элемента <xsl:when>. Приведенный фрагмент может рассматриваться как эквивалент следующей конструкции из процедурного программирования:

```
int y;
if (x > 7)
  y = 13;
else
  y = 15;
```

А это настоящие переменные?

Несмотря на свое название, переменные XSLT не являются переменными в смысле, традиционном для процедурных языков программирования, таких как C++ или Java. Помните, о чем мы говорили ранее: одной из целей проектирования языков таблиц стилей является предотвращение побочных эффектов при выполнении. Однако изменение

значения переменной относится к числу наиболее стандартных побочных эффектов в большинстве процедурных языков. Если бы результат обработки стилей зависел от изменяющихся значений разных переменных, ядро таблиц стилей было бы вынуждено обрабатывать шаблоны в определенном порядке.

Переменные XSLT больше напоминают переменные в традиционном математическом смысле: вы определяете функцию `square(x)`, которая возвращает значение числа (представленного переменной `x`), умноженного само на себя. Другими словами, `square(2.5)` возвращает `6.25`. В таком контексте `x` может быть любым числом, но функция `square` не может изменить значения `x`.

К этой концепции привыкаешь не сразу, но у вас получится. Поверьте мне.

Область видимости переменной

Область видимости элемента `<xsl:variable>` ограничивается по тому элементу, который его содержит. Если элемент `<xsl:variable>` является элементом верхнего уровня (т. е. его родителем является `<xsl:stylesheet>`), то переменная считается глобальной и доступна в любой точке таблицы стилей. Элемент `<xsl:variable>` также может использоваться в `<xsl:template>` для локального переопределения глобальной переменной.

Рекурсия

Написание таблицы стилей XSLT отличается от программирования на других языках. Если до сих пор вы этого не поняли, то наверняка убедитесь в этом сейчас. Глава завершается парой примеров, показывающих, как при помощи рекурсии решаются проблемы, типичные для процедурных языков программирования. Мы также рассмотрим некоторые новые возможности XSLT 2.0 и XPath 2.0, позволяющие обойтись без рекурсии в некоторых ситуациях.

Реализация функции замены строк

Чтобы продемонстрировать применение рекурсии для решения практических задач, мы напишем функцию замены строк. Такая функция может пригодиться, например, для экранирования некоторых символов или подстрок в выходных данных. Создаваемая таблица стилей преобразует документ XML в набор команд SQL для выполнения в приглашении командной строки Windows. В ходе преобразования должны быть выполнены следующие замены:

Поставить знак ^ перед каждым амперсандом (`&`)

В командной строке Windows NT и Windows 2000 амперсанд обозначает завершение одной и начало другой команды. Например,

следующая команда создает новый каталог с именем *xslt* и переходит в созданный каталог:

```
mkdir xslt & chdir xslt
```

Если создаваемая команда SQL содержит амперсанды, необходимо экранировать эти символы, чтобы они обрабатывались как литералы, а не как операторы. Например, если запись, вставляемая в базу данных, содержит строку `Jones & Son`, перед выполнением команды SQL необходимо привести ее к виду `Jones ^& Son`.

Поставить знак ^ перед каждой вертикальной чертой (|)

Вертикальная черта в семействе Windows используется для конвейерной передачи данных. Если мы хотим, чтобы этот символ интерпретировался как литеральный текст, а не как оператор, его необходимо экранировать.

Заменить каждый одиночный апостроф (') двойным апострофом ('')

Требование используемой СУБД.

Процедурная реализация

В нашем шаблоне будет уместно использовать три функции: `concat()`, `substring-before()` и `substring-after()`. Замена амперсанда последовательностью `^&` осуществляется так:

```
<xsl:value-of select="concat(substring-before(., '&amp;'), '^&amp;',
                             substring-after(., '&amp;'))"/>
```

У такого решения есть очевидный недостаток: оно заменяет только первое вхождение амперсанда. Если строка содержит два, три или триста амперсандов, метод придется вызывать по одному разу для каждого вхождения. Из-за особенностей переменных XSLT мы не сможем действовать так, как в процедурном языке:

```
private static String strChange(String string, String from, String to)
{
  String before = "", after = "";
  int    index;

  index = string.indexOf(from);
  while (index >= 0)
  {
    before = string.substring(0, index);
    after = string.substring(index + from.length());
    string = before + to + after;

    index = string.indexOf(from, index + to.length());
  }

  return string;
}
```

XSLT не имеет простого механизма перебора символов строки, поэтому вместо этого мы воспользуемся рекурсией.

Рекурсивная реализация

Рекурсивная реализация функции замены строк строится на основе следующего алгоритма:

- Если вся строка *содержит* заменяемую подстроку, выполнить следующие действия:
  1. Вернуть первую часть всей строки – все, что предшествует заменяемой строке.
  2. Вернуть заменяющую подстроку.
  3. Вернуть результат вызова функции для завершающей части исходной строки – всех символов за первым вхождением заменяемой подстроки. *На этой стадии проявляется рекурсивная природа реализации*.
- Если вся строка *не содержит* заменяемую подстроку, просто вернуть всю строку.

Если заменяемая подстрока присутствует во входной строке, мы вызываем функцию замены для оставшейся части строки. Ключевым моментом (как и во всех рекурсивных функциях) является *условие выхода* – условие, при котором рекурсивный вызов не выполняется. Рано или поздно рекурсивная функция будет вызвана со строкой, которая не содержит заменяемой подстроки.

Реализация на псевдокоде выглядит так:

```
replaceSubstring(originalString, substring, replacementString)
{
  if (contains(originalString, substring))
  {
    return
      (substring-before(originalString, substring) +
       replacementString +
       replaceSubstring(substring-after(originalString, substring),
                        substring, replacementString));
  }
  else
    return originalString;
}
```

В рекурсивной реализации функция вызывает сама себя, пока она содержит хотя бы одно вхождение заменяемой подстроки. Каждый раз, когда функция вызывает сама себя, параметр `originalString` частично уменьшается до тех пор, пока не будет обработана вся строка. Далее приводится полный листинг таблицы стилей:

```
<?xml version="1.0"?>
<!-- string_replace-1_0.xsl -->
```

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:apply-templates select="ul/li"/>
  </xsl:template>

  <xsl:template match="li">
    <xsl:variable name="single-quote">
      <xsl:text>&apos;</xsl:text>
    </xsl:variable>
    <xsl:variable name="two-quotes">
      <xsl:text>&apos;&apos;</xsl:text>
    </xsl:variable>

    <xsl:variable name="sub1">
      <xsl:call-template name="replace-substring">
        <xsl:with-param name="original" select="."/>
        <xsl:with-param name="substring" select="'&amp;'"/>
        <xsl:with-param name="replacement" select="'^&amp;'"/>
      </xsl:call-template>
    </xsl:variable>

    <xsl:variable name="sub2">
      <xsl:call-template name="replace-substring">
        <xsl:with-param name="original" select="$sub1"/>
        <xsl:with-param name="substring" select="'|'"/>
        <xsl:with-param name="replacement" select="'^|'"/>
      </xsl:call-template>
    </xsl:variable>

    <xsl:call-template name="replace-substring">
      <xsl:with-param name="original" select="$sub2"/>
      <xsl:with-param name="substring" select="$single-quote"/>
      <xsl:with-param name="replacement" select="$two-quotes"/>
    </xsl:call-template>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

  <xsl:template name="replace-substring">
    <xsl:param name="original" />
    <xsl:param name="substring" />
    <xsl:param name="replacement" />
    <xsl:choose>
      <xsl:when test="contains($original, $substring)">
        <xsl:value-of
          select="substring-before($original, $substring)" />
        <xsl:value-of select="$replacement" />
        <xsl:call-template name="replace-substring">
          <xsl:with-param name="original"
            select="substring-after($original, $substring)" />
          <xsl:with-param
            name="substring" select="$substring" />
```

```
          <xsl:with-param
            name="replacement" select="$replacement" />
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$original" />
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

</xsl:stylesheet>
```

Переменная `$sub1` создается посредством замены всех амперсандов в исходном тексте последовательностью `^&`. Переменная `$sub2` создается посредством замены всех вертикальных черт в исходном тексте последовательностью `^|`. Наконец, значение `$sub2` используется при третьем вызове шаблона замены, удваивающем все одиночные апострофы. Обратите внимание: третий вызов не включен в элемент `<xsl:variable>`, поэтому его результат записывается в выходной документ.

Для следующего входного документа XML:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- testlines.xml -->
<ul>
  <li>This is a test &amp;amp; I hope it works | fails gracefully</li>
  <li>Some techniques are simpler &amp;amp; easier than recursion</li>
  <li>Will I enjoy next Tuesday's meeting?</li>
</ul>
```

Рекурсивный шаблон возвращает следующий результат:

```
This is a test ^& I hope it works ^| fails gracefully
Some techniques are simpler ^& easier than recursion
Will I enjoy next Tuesday''s meeting?
```

К такому стилю программирования привыкаешь не сразу, но обычно на базе рекурсии удается решить практически любую задачу. Приведенный пример хорошо демонстрирует приемы, описанные в этой главе, в том числе команды передачи управления, переменные, вызов шаблонов по имени и передачу параметров.

[2.0] Использование функции XPath 2.0 replace() для предотвращения рекурсии

Операции со строками принадлежат к числу стандартных задач преобразования документов, поэтому в XPath 2.0 появилась очень полезная функция `replace()`. Она позволяет задать исходную строку, искомую строку и ту строку, которой она должна быть заменена. Возвращаясь к предыдущему примеру, мы хотим выполнить в тексте три замены:

- Каждый амперсанд (`&`) должен быть заменен последовательностью `^&`

- Каждая вертикальная черта (|) должна быть заменена последовательностью ^|
- Каждый апостроф (') должен быть заменен двумя апострофами ('')

Таблица стилей для выполнения этих операций выглядит так:

```
<?xml version="1.0"?>
<!-- string_replace-2_0.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:apply-templates select="ul/li"/>
  </xsl:template>

  <xsl:template match="li">
    <xsl:variable name="sub1" select="replace(., '&amp;', '^&amp;')"/>
    <xsl:variable name="sub2" select="replace($sub1, '\|', '^|')"/>
    <xsl:value-of select='replace($sub2, "&apos;", "&apos;&apos;")'/>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Результат будет таким же, как у гораздо более длинной таблицы стилей XSLT 1.0:

```
This is a test ^& I hope it works ^| fails gracefully
Some techniques are simpler ^& easier than recursion
Will I enjoy next Tuesday''s meeting?
```

Этот пример гораздо проще рекурсии, использованной в таблице стилей XSLT 1.0. Обе таблицы выдают одинаковые результаты, но код примера XSLT 2.0 намного понятнее.

Краткое замечание по поводу синтаксиса экранирования апострофов в таблице стилей XSLT 2.0 (когда один апостроф заменяется двумя): в XPath 2.0 апострофы могут экранироваться удвоением в выражениях XPath, так что вызов функции `replace()` может быть записан в следующем виде:

```
<xsl:value-of select="replace($sub2, '''', '''''')"/>
```

Синтаксис весьма запутанный, но он работает.

Имитация цикла for

Ранее уже подчеркивалось, что элемент `xsl:for-each` не является аналогом цикла `for`; это всего лишь итератор для перебора группы узлов. Впрочем, если вам потребуется реализовать цикл `for`, это тоже возможно. (Однако приготовьтесь к использованию рекурсии.)

Структура шаблона

Мы хотим создать именованный шаблон, который получает аргументы и имитирует функциональность цикла `for`. Традиционные циклы `for` обладают рядом стандартных компонентов:

Одна или несколько команд инициализации

Команды инициализации обрабатываются до начала цикла `for`. Как правило, команды инициализации относятся к *управляющей (индексной) переменной*, определяющей признак продолжения цикла.

Команда приращения

Команда, указывающая, каким образом управляющая переменная должна обновляться после каждой итерации цикла.

Логическое выражение

Если выражение истинно, цикл продолжается; если оно становится ложным, цикл завершается.

Возьмем пример из мира Java и C++:

```
for (int i=0; i<length; i++)
```

В этом невероятно оригинальном примере используется команда инициализации `i=0`. Управляющей переменной (т. е. переменной, от значения которой зависит продолжение или прерывание цикла) является переменная `i`. Продолжение цикла определяется логическим выражением `i<length`; наконец, управляющая переменная изменяется командой приращения `i++`.

В своей реализации мы сделаем несколько упрощающих предположений (дорогой читатель, ничто не мешает вам сделать свою реализацию настолько сложной, насколько вы сочтете нужным):

- Вместо команды инициализации вызывающая сторона должна задать значение локальной переменной `i` при вызове обработчика цикла `for`. Это значение передается в виде параметра, чтобы оно могло вычисляться в выражениях XPath.
- Вместо определения команд приращения вида `i++` вызывающая сторона должна задать значение локальной переменной `increment`. По умолчанию эта переменная равна `1`, однако она может содержать произвольное отрицательное или положительное целое. Значение переменной прибавляется к текущему значению `i` после каждой итерации цикла.
- Вместо определения произвольного логического выражения вызывающая сторона должна передавать два параметра, `operator` и `testValue`. Допустимые значения переменной `operator`: `=`, `<` (кодируется в виде `<`), `>` (кодируется в виде `>`), `!=`, `<=` (кодируется в виде `<=`) и `>=` (кодируется в виде `>=`). Выбор такого решения объясняется тем, что мы не можем потребовать у процессора XSLT, что-

бы он вычислил литеральное условие вида `i<length` как часть таблицы стилей.

Реализация

В нашей таблице стилей определяются четыре глобальные переменные:

```
<xsl:param name="i"         select="1"/>
<xsl:param name="increment" select="1"/>
<xsl:param name="operator"  select="'<='"/>
<xsl:param name="testValue" select="10"/>
```

Определяемые значения по умолчанию соответствуют команде C++ или Java `for (i = 1; i<10; i++)`. Также имеется шаблон `match="/"`, активизирующий обработку цикла `for`:

```
<xsl:template match="/">
  <xsl:call-template name="for-loop">
    <xsl:with-param name="i"         select="$i"/>
    <xsl:with-param name="increment" select="$increment"/>
    <xsl:with-param name="operator"  select="$operator"/>
    <xsl:with-param name="testValue" select="$testValue"/>
  </xsl:call-template>
</xsl:template>
```

Обработка шаблона цикла `for` должна начинаться с проверки истинности условия. Для этого мы вычисляем логическое значение с несколькими элементами `<xsl:when>`, каждый из которых выглядит примерно так:

```
  <xsl:variable name="testPassed">
    <xsl:choose>
      <xsl:when test="$operator = '!='">
        <xsl:if test="$i != $testValue">
          <xsl:text>true</xsl:text>
        </xsl:if>
      </xsl:when>
      ...
  </xsl:variable>
```

Если переменная `$testPassed` равна `true`, цикл `for` снова вызывает сам себя. Перед инструкцией `<xsl:call-template>` размещается вся необходимая логика цикла. В нашем примере текущее значение `$i` просто записывается в выходной поток.

Полный код примера

Полный код таблицы стилей выглядит так:

```
<?xml version="1.0"?>
<!-- for-loop.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>
```

```
<xsl:param name="i" select="1"/>
<xsl:param name="increment" select="1"/>
<xsl:param name="operator" select="'&lt;='"/>
<xsl:param name="testValue" select="10"/>

<xsl:template match="/">
  <xsl:call-template name="for-loop">
    <xsl:with-param name="i"         select="$i"/>
    <xsl:with-param name="increment" select="$increment"/>
    <xsl:with-param name="operator"  select="$operator"/>
    <xsl:with-param name="testValue" select="$testValue"/>
  </xsl:call-template>
</xsl:template>

<xsl:template name="for-loop">
  <xsl:param name="i"/>
  <xsl:param name="increment"/>
  <xsl:param name="operator"/>
  <xsl:param name="testValue"/>

  <xsl:variable name="testPassed">
    <xsl:choose>
      <xsl:when test="$operator = '!='">
        <xsl:if test="$i != $testValue">
          <xsl:text>true</xsl:text>
        </xsl:if>
      </xsl:when>
      <xsl:when test="$operator = '&lt;='">
        <xsl:if test="$i &lt;= $testValue">
          <xsl:text>true</xsl:text>
        </xsl:if>
      </xsl:when>
      <xsl:when test="$operator = '&gt;='">
        <xsl:if test="$i &gt;= $testValue">
          <xsl:text>true</xsl:text>
        </xsl:if>
      </xsl:when>
      <xsl:when test="$operator = '='">
        <xsl:if test="$i = $testValue">
          <xsl:text>true</xsl:text>
        </xsl:if>
      </xsl:when>
      <xsl:when test="$operator = '&lt;'">
        <xsl:if test="$i &lt; $testValue">
          <xsl:text>true</xsl:text>
        </xsl:if>
      </xsl:when>
      <xsl:when test="$operator = '&gt;'">
        <xsl:if test="$i &gt; $testValue">
          <xsl:text>true</xsl:text>
        </xsl:if>
      </xsl:when>
```

```
        <xsl:otherwise>
          <xsl:message terminate="yes">
            <xsl:text>Sorry, the for-loop emulator only </xsl:text>
            <xsl:text>handles six operators &#xA;</xsl:text>
            <xsl:text>(&lt; | &gt; | = | &lt;= | &gt;= | !=).  </xsl:text>
            <xsl:text>The value </xsl:text>
            <xsl:value-of select="$operator"/>
            <xsl:text> is not allowed.&#xA;</xsl:text>
          </xsl:message>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>

    <xsl:if test="$testPassed='true'">
      <!-- Здесь размещается произвольная логика тела цикла.           -->
      <!-- В нашем примере цикл просто выводит текст в выходной поток. -->
      <xsl:text>Value of i=</xsl:text>
      <xsl:value-of select="$i"/>
      <xsl:text>&#xA;</xsl:text>

      <!-- Здесь логика тела цикла заканчивается; не изменяйте         -->
      <!-- оставшуюся часть шаблона!                                   -->

      <!-- А теперь важная часть: приращение управляющей переменной.   -->
      <!-- Обратите внимание: мы передаем измененное значение,         -->
      <!-- а не изменяем саму переменную.                              -->

      <xsl:call-template name="for-loop">
        <xsl:with-param name="i"         select="$i + $increment"/>
        <xsl:with-param name="increment" select="$increment"/>
        <xsl:with-param name="operator"  select="$operator"/>
        <xsl:with-param name="testValue" select="$testValue"/>
      </xsl:call-template>
    </xsl:if>
  </xsl:template>

</xsl:stylesheet>
```

Со значениями параметров по умолчанию таблица стилей выводит ожидаемый результат:

```
Value of i=1
Value of i=2
Value of i=3
Value of i=4
Value of i=5
Value of i=6
Value of i=7
Value of i=8
Value of i=9
Value of i=10
```

С параметрами `i=10`, `increment="-1"`, `operator=">="` и `testValue=0` мы получим следующий результат:

```
Value of i=10
Value of i=9
Value of i=8
Value of i=7
Value of i=6
Value of i=5
Value of i=4
Value of i=3
Value of i=2
Value of i=1
Value of i=0
```

Кавычки, в которые заключаются значения `increment` и `operator`, необходимы при передаче этих значений процессору XSLT из командной строки. В завершение приведу результаты для `i=10`, `increment="-2"`, `operator=">"` и `testValue=0`:

```
Value of i=10
Value of i=8
Value of i=6
Value of i=4
Value of i=2
```

Если вы хотите изменить цикл `for` так, чтобы он делал что-то более полезное, вставьте свой код между следующими комментариями:

```
<!-- Здесь логика тела цикла заканчивается; не изменяйте        -->
<!-- оставшуюся часть шаблона!                                  -->

<!-- А теперь важная часть: приращение управляющей переменной.  -->
<!-- Обратите внимание: мы передаем измененное значение,        -->
<!-- а не изменяем саму переменную.                             -->
```

Итоги

Весьма внушительный объем материала для одной главы, не правда ли? Мы рассмотрели все базовые элементы, необходимые для реализации программной логики в таблицах стилей. В частности, были описаны некоторые общие особенности XSLT и других языков программирования, о которых следует знать; что еще важнее, мы обсудили отличия XSLT от процедурного кода, который вам, вероятно, доводилось писать ранее. Особенно непривычно смотрятся рекурсия и переменные, которые не изменяются. Несмотря на определенные трудности с освоением, большинство повседневных задач решается по аналогии с примерами, приведенными в этой главе. Разобравшись с базовыми элементами, мы переходим к связыванию и перекрестным ссылкам. Следующая глава посвящена различным способам установления связей между различными частями документов XML.

6

Связывание и перекрестные ссылки

Чем бы вы ни занимались – построением веб-сайтов, публикацией книг, обработкой заказов на базе XML – скорее всего, многие информационные объекты в генерируемых документах будут содержать ссылки на другую информацию. В этой главе рассматриваются некоторые способы связывания элементов XML. Здесь будут описаны три механизма:

- Использование типов данных XML `ID`, `IDREF` и `IDREFS`
- Расширенное связывание с применением функции `key()`
- Генерирование ссылок в неструктурированных документах

Типы данных XML ID, IDREF и IDREFS

Наше знакомство со ссылками начнется с функции XPath `id()`. Эта полезная функция позволяет найти элемент, обладающий атрибутом `ID` с заданным значением.

Типы данных и как они работают

К числу базовых типов данных, поддерживаемых в XML DTD (Document Type Definition) и XML Schema, относятся типы `ID`, `IDREF` и `IDREFS`. Типы данных `ID` и `IDREF` подчиняются двум правилам:

- Все атрибуты типа данных `ID` должны быть уникальными
- Любое значение типа данных `IDREF` должно соответствовать значению атрибута типа данных `ID`, находящемуся где-то в документе

Атрибут с типом данных `IDREFS` содержит одно или несколько значений, разделенных пробелами, каждое из которых должно соответствовать значению `ID`, находящемуся где-то в документе. Тип данных

IDREFS, как следует из его названия, представляет собой список значений IDREF.

Простой фрагмент DTD с использованием типов данных ID и IDREF:

```
<?xml version="1.0"?>
<!-- parts-list1.xml -->
<!DOCTYPE parts-list [
<!ELEMENT parts-list       (component+, part+)>
<!ELEMENT component        (name, partref+)>
<!ATTLIST component        component-id ID #REQUIRED>

<!ELEMENT name             (#PCDATA)>

<!ELEMENT partref          EMPTY>
<!ATTLIST partref          refid IDREF #REQUIRED>

<!ELEMENT part             (name)>
<!ATTLIST part             part-id ID #REQUIRED>
]>

<parts-list>
...
</parts-list>
```

А вот как выглядит определение этого типа документа в XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- parts-list.xsd -->
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="parts-list">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="component" minOccurs="1" maxOccurs="unbounded"/>
        <xs:element ref="part" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="component">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="partref" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="component-id" type="xs:ID" use="required"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="part">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
```

```
      <xs:attribute name="part-id" type="xs:ID" use="required"/>
    </xs:complexType>
  </xs:element>

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

  <xs:element name="partref">
    <xs:complexType>
      <xs:attribute name="refid" type="xs:IDREF" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Приведенные описания DTD и XML Schema семантически эквивалентны, поэтому для нас несущественно, какой из двух документов будет использоваться для проверки файлов XML. (По своим выразительным возможностям XML Schema значительно превосходит DTD, но в данном случае два документа полностью совпадают по смыслу.) В них встречаются примеры использования `ID` и `IDREF`. Мы также рассмотрим реже используемый тип данных `IDREFS`.

Учтите, что в документе XML должны быть проверены все атрибуты со значениями типов данных `ID`, `IDREF` и `IDREFS`. Позднее эта тема будет изложена более подробно, но если вам не терпится, переходите к разделу «[2.0] Функция idref()» далее в этой главе.

Попробуем в двух словах описать, как работает документ со списком деталей: действительный элемент `<part-list>` содержит один или несколько элементов `<component>`, за которыми следует один или несколько элементов `<part>`. Элементы `<component>` и `<part>` должны иметь атрибут типа `ID` (`component-id` или `part-id` соответственно). Имеется также элемент `<partref>` с обязательным атрибутом типа `IDREF`; значение этого атрибута с именем `refid` должно совпадать со значением элемента `ID` где-то в списке деталей.

Эти четко структурированные документы будут использоваться в нашей первой попытке создания ссылок в документах.

Связывание частей документа XML

Для демонстрации связывания мы воспользуемся документом, определяющим несколько элементов `<component>` и `<part>`. Каждый компонент (элемент `<component>`) использует некоторое количество деталей (элемент `<part>`). Документ XML содержит значения типов `ID` и `IDREF`, которые позволяют связать ссылками разные части документа.

Вот как выглядит наш документ:

```
<?xml version="1.0"?>
<!-- parts-list1.xml -->
<!DOCTYPE parts-list [
...
```

```
]>

<parts-list>
  <component component-id="C28392-33-TT">
    <name>Turnip Twaddler</name>
    <partref refid="P81952-26-PK"/>
    <partref refid="P86679-52-SP"/>
    <partref refid="P81472-68-FD"/>
    <partref refid="P88107-39-GT"/>
  </component>
...
  <component component-id="C28772-63-OB">
    <name>Olive Bruiser</name>
    <partref refid="P80228-21-PT"/>
    <partref refid="P82387-85-PA"/>
  </component>

  <part part-id="P80228-21-PT">
    <name>Pitter</name>
  </part>
...
  <part part-id="P86994-25-RC">
    <name>Ribbon Curler</name>
  </part>
</parts-list>
```

Наша первая задача – последовательно обратиться к каждому элементу `<component>` и получить имена используемых им элементов `<part>`. Каждый атрибут `refid` каждого элемента `<partref>` соответствует атрибуту `id` элемента `<part>`.

Таблица стилей с использованием функции id()

Для начала посмотрим, что мы хотим получить. Наша задача – создать простой текстовый документ с перечнем названий деталей по каждому компоненту, который выглядит примерно так:

```
Here is a test of the id() function:

  Turnip Twaddler (component #C28392-33-TT) uses these parts:
    Spanner
    Feather Duster
    Grommet
    Paring Knife

  Prawn Goader (component #C28813-70-PG) uses these parts:
    Paring Knife
    Mucilage
    Ribbon Curler

...

  Olive Bruiser (component #C28772-63-OB) uses these parts:
    Pitter
    Patter
```

Таблица стилей, генерирующая эти результаты, достаточно тривиальна:

```
<?xml version="1.0"?>
<!-- id1.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Here is a test of the id() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:for-each select="/parts-list/component">
      <xsl:text>&#xA;  </xsl:text>
      <xsl:value-of select="name"/>
      <xsl:text> (component #</xsl:text>
      <xsl:value-of select="@component-id"/>
      <xsl:text>) uses these parts:&#xA;    </xsl:text>
      <xsl:for-each select="id(partref/@refid)">
        <xsl:value-of select="name"/>
        <xsl:text>&#xA;    </xsl:text>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Мы вызываем функцию `id()`, которая возвращает набор узлов/последовательность всех узлов, соответствующих всем атрибутам `refid` элементов `<partref>` в каждом компоненте. Каждый объект в наборе узлов представляет собой элемент `<part>`, так что выражение XPath `select="name"` возвращает название детали (текст дочернего элемента `<name>`).

У обеих функций, `id()` и `idref()`, имеется версия с двумя аргументами. Второй аргумент обеих функций определяет узел. Параметр приказывает процессору XSLT выполнять поиск в документе, содержащем узел (вместо документа, содержащего контекстный объект). Скорее всего, такая возможность вам не понадобится, но она существует.

Прежде чем переходить к более сложным примерам, рассмотрим несколько иной документ XML, в котором вместо `IDREF` используются атрибуты типа `IDREFS`:

```
<?xml version="1.0"?>
<!-- parts-list2.xml -->
<!DOCTYPE parts-list [
<!ELEMENT parts-list       (component+, part+)>

<!ELEMENT component        (name, partref)>
<!ATTLIST component        component-id ID #REQUIRED>

<!ELEMENT name             (#PCDATA)>

<!ELEMENT partref          EMPTY>
```

```
<!ATTLIST partref          refid IDREFS #REQUIRED>

<!ELEMENT part             (name) >
<!ATTLIST part             part-id ID #REQUIRED>
]>

<parts-list>
  <component component-id="C28392-33-TT">
    <name>Turnip Twaddler</name>
    <partref
      refid="P81952-26-PK P86679-52-SP P81472-68-FD P88107-39-GT"/>
  </component>
  <component component-id="C28813-70-PG">
    <name>Prawn Goader</name>
    <partref refid="P81952-26-PK P80499-43-MC P86994-25-RC"/>
  </component>
...
</parts-list>
```

Несмотря на изменившуюся структуру документа XML, его можно обработать той же таблицей стилей. *Мы получим те же результаты, хотя тип данных атрибута изменился.* При вызове функции `id()` с аргументом вида `P81952-PK P86679-52-SP...` функция `id()` интерпретирует каждую строку, отделенную пробелами, как отдельный идентификатор. Значение каждого атрибута `ID` должно быть действительным именем XML, а это означает, что оно не может содержать пробелы; благодаря этому обстоятельству описанная схема работает. (О действительных именах XML мы поговорим чуть позже.)

Итак, мы написали таблицу стилей, которая переходит от `IDREF` к элементу с конкретным значением `ID`. Теперь мы напишем таблицу, которая переходит от значения `ID` ко всем ссылкам на него. Иначе говоря, таблица должна выводить список всех деталей в документе, а затем перечислять все компоненты, в которых эта деталь используется. Основные проблемы возникают с написанием выражения XPath; как по значению `ID` элемента `<part>` определить все элементы `<component>`, содержащие элемент `<partref>` с атрибутом `refid`? Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- id2.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Here is a test of the id() </xsl:text>
    <xsl:text>function in reverse:&#xA;</xsl:text>

    <xsl:for-each select="/parts-list/part">
      <xsl:text>&#xA;  </xsl:text>
      <xsl:value-of select="name"/>
      <xsl:text> (part #</xsl:text>
```

```
      <xsl:value-of select="@part-id"/>
      <xsl:text>) is used in these products:&#xA;</xsl:text>
      <xsl:for-each
        select="/parts-list/component
                [partref/@refid=current()/@part-id]">
        <xsl:value-of select="name"/>
        <xsl:if test="position() != last()">
          <xsl:text>&#xA;    </xsl:text>
        </xsl:if>
      </xsl:for-each>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Выражение XPath в элементе `<xsl:for-each>` стало более сложным; давайте посмотрим на него внимательнее.

```
/parts-list/component[partref/@refid=current()/@part-id]
```

Выражение возвращает все элементы `<component>`, для которых предикатное выражение истинно. Предикатное выражение определяет компонент, содержащий как минимум один дочерний элемент `<partref>`, у которого атрибут `refid` соответствует атрибуту `part-id` текущего узла.

Предикат сравнивает два значения, хотя внешне он и выглядит как путь доступа, состоящий из трех частей. Одна из частей (`partref/@refid`) определяет набор атрибутов `refid` всех элементов `<partref>` в заданном `<component>`. Другое значение (`current()/@part-id`) является значением атрибута `part-id()` текущего узла. Каждое значение в предикате относится к элементу `<component>`; функция `current()` возвращает текущий обрабатываемый элемент `<part>`. Элемент `<part>` выбирается первым элементом `<xsl:for-each>` в шаблоне.

Результат применения таблицы стилей:

```
Here is a test of the id() function in reverse:

  Pitter (part #P80228-21-PT) is used in these products:
    Olive Bruiser

  Patter (part #P82387-85-PA) is used in these products:
    Olive Bruiser

  Spanner (part #P86679-52-SP) is used in these products:
    Turnip Twaddler
    Clam Teaser
    Lemon Snubber

  Feather Duster (part #P81472-68-FD) is used in these products:
    Turnip Twaddler
    Clam Teaser
    Cucumber Decorating Kit
...
```

Во внутреннем элементе <xsl:for-each> выражение XPath name возвращает название для текущего элемента <component>. Внутренний элемент <xsl:for-each> также можно было записать следующим образом:

```
<xsl:for-each
  select="/parts-list/component/partref
          [@refid=current()/@part-id]">
  <xsl:value-of select="../name"/>
```

Результат остается неизменным, но выражение для выбора имени компонента слегка усложняется. Атрибут select элемента <xsl:for-each> возвращает набор узлов элементов <partref>, поэтому для получения названия компонента мы должны использовать запись ../name. Если вам приходится записывать много сложных выражений в элементе <xsl:for-each>, посмотрите, нельзя ли переписать выражение XPath в элементе <for-each> для упрощения таблицы стилей.

[2.0] Функция idref()

В предыдущей таблице стилей «обратное» использование функции id() (для перехода от заданного атрибута ID к элементам, ссылающимся на него) было довольно неудобным. Данная задача является весьма стандартной, поэтому в XSLT 2.0 появилась функция idref(). По заданному значению ID функция idref() возвращает все элементы, ссылающиеся на него. Далее приводится простая таблица стилей для работы со списком деталей:

```
<?xml version="1.0"?>
<!-- idref.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Here is a test of the idref() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:for-each select="/parts-list/part">
      <xsl:text>&#xA;  </xsl:text>
      <xsl:value-of select="name"/>
      <xsl:text> (part #</xsl:text>
      <xsl:value-of select="@part-id"/>
      <xsl:text>) is used in these products:&#xA;</xsl:text>
      <xsl:value-of select="idref(@part-id)/../../name"
        separator="&#xA;"/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

В этой таблице стилей все делается в одном элементе <xs:value>. Обратите внимание: функция idref() возвращает совпадающие *атрибуты*,

поэтому мы используем выражение `idref(@part-id)/../../name` для получения имени компонента. Родителем атрибута является элемент `<partref>`, а его родителем – элемент `<component>`. В данном случае нас интересует дочерний элемент `<name>` элемента `<component>`.

Наконец, новая функция `idref()` работает с типом данных `IDREFS`, как и `id()`.

Генерирование ссылок в документах HTML

Прежде чем переходить к следующей теме, рассмотрим более сложную таблицу стилей, генерирующую код HTML. Требуется перечислить все компоненты и детали из документа и создать гиперссылки между ними. Таким образом, сначала мы строим список всех компонентов и используемых ими деталей; каждое название детали оформляем как ссылку на описание детали. И то же самое делаем при перечислении деталей и компонентов, в которых они используются. Исходный документ слегка изменен для следующих примеров:

```
<?xml version="1.0"?>
<!-- parts-list3.xml -->
<!DOCTYPE parts-list [
<!ELEMENT parts-list       (component+, part+, supplier+)>

<!ELEMENT component        (name, partref+, description)>
<!ATTLIST component        component-id ID #REQUIRED>

<!ELEMENT name             (#PCDATA)>

<!ELEMENT partref          EMPTY>
<!ATTLIST partref          refid IDREF #REQUIRED>

<!ELEMENT part             (name, description)>
<!ATTLIST part             part-id ID #REQUIRED
                           supplier CDATA #REQUIRED>

<!ELEMENT description      (#PCDATA|partref)*>

<!ELEMENT supplier         (name)>
<!ATTLIST supplier         country CDATA #REQUIRED
                           vendor-id CDATA #REQUIRED>
]>

<parts-list>
  <component component-id="C28392-33-TT">
    <name>Turnip Twaddler</name>
    <partref refid="P81952-26-PK"/>
    <partref refid="P86679-52-SP"/>
    <partref refid="P81472-68-FD"/>
    <partref refid="P88107-39-GT"/>
    <description>
      If you've got turnips to twaddle, this is the tool for you!
      Comes with a <partref refid="P81472-68-FD"/>.
    </description>
  </component>
```

```
  <component component-id="C28100-38-CT">
    <name>Clam Teaser</name>
    <partref refid="P81472-68-FD"/>
    <partref refid="P86994-25-RC"/>
    <partref refid="P86679-52-SP"/>
    <description>
      Everyone knows they're proverbially happy, but what to
      do with a shy clam?  Bring recalcitrant mollusks out of
      their shells with this entertaining gadget.  Includes a
      festive <partref refid="P86994-25-RC"/>.
    </description>
  </component>

...

  <part part-id="P80228-21-PT" supplier="4839">
    <name>Pitter</name>
    <description>
      Removes pits from olives and cherries in no time at all.
    </description>
  </part>
  <part part-id="P82387-85-PA" supplier="2983">
    <name>Patter</name>
    <description>
      We're not sure what these things do, but people seem
      to like 'em.
    </description>
  </part>

...

  <supplier country="Great Britain" vendor-id="4839">
    <name>Acme Products, Inc.</name>
  </supplier>
  <supplier country="Germany" vendor-id="2983">
    <name>Deutschland Excelsior Gmbh</name>
  </supplier>
  <supplier country="Great Britain" vendor-id="5910">
    <name>Unlimited Spanners Ltd.</name>
  </supplier>
</parts-list>
```

Между этим документом и прототипом существует пара различий. Во-первых, в каждый элемент `<component>` и `<part>` включен элемент `<description>` с описанием. Положение усложняется тем, что некоторые описания содержат элемент `<partref>`. Обратите внимание: элемент `<partref>` не содержит текста. Это означает, что название детали определяется только в одном месте; если изменить название детали, оно автоматически обновится во всех местах использования. Во-вторых, в документе появились элементы `<supplier>` с информацией о поставщике, а в каждую деталь добавлены атрибуты `country` и `vendor-id`.

О том, как они используются, рассказано ниже – при обсуждении ключей и ключевых функций.

Для построения нужного документа в формате HTML необходимо создать точки связывания. Описание каждого компонента и каждой детали должно быть снабжено якорем HTML (`<a name="...">`), чтобы на каждое описание можно было создать ссылку. Поскольку детали и компоненты уже обладают уникальными идентификаторами, мы используем значения этих идентификаторов в качестве имен точек связывания. Таким образом, мы знаем, как создать точку связывания для каждого компонента и детали и как создать ссылку на нужную деталь или компонент.

Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- id-html.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Our Catalog</title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>Our Catalog</h1>
        <p>Here's a look at everything in our catalog:</p>
        <h2 style="background: #66FF66;">Components</h2>
        <xsl:apply-templates select="/parts-list/component"/>
        <h2 style="background: #6666FF;">Parts</h2>
        <xsl:apply-templates select="/parts-list/part"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="component">
    <a name="{@component-id}"/>
    <h3>
      <xsl:value-of select="name"/>
    </h3>
    <p>
      <xsl:apply-templates select="description"/>
    </p>
    <p>
      <xsl:value-of select="name"/>
      <xsl:text> uses these parts:</xsl:text>
    </p>
    <ul>
      <xsl:for-each select="partref">
        <li>
```

```
        <xsl:apply-templates select="."/>
      </li>
    </xsl:for-each>
  </ul>
</xsl:template>

<xsl:template match="description">
  <xsl:apply-templates select="*|text()"/>
</xsl:template>

<xsl:template match="partref">
  <a href="{concat('#', @refid)}">
    <xsl:value-of select="id(@refid)/name"/>
  </a>
</xsl:template>

<xsl:template match="part">
  <a name="{@part-id}"/>
  <h3>
    <xsl:value-of select="name"/>
  </h3>
  <p>
    <xsl:apply-templates select="description"/>
  </p>
  <p>
    <xsl:value-of select="name"/>
    <xsl:text> is used in these components:</xsl:text>
  </p>
  <ul>
    <xsl:for-each select="/parts-list/component
                          [partref/@refid=current()/@part-id]">
      <li>
        <a href="{concat('#', @component-id)}">
          <xsl:value-of select="name"/>
        </a>
      </li>
    </xsl:for-each>
  </ul>
</xsl:template>

</xsl:stylesheet>
```

А вот как создаются точки связывания:

```
<a name="{@part-id}"/>
```

Эта конструкция генерирует разметку HTML `<a name="C28392-33-TT"/>` **для первого элемента** `<component>` **в документе. Для создания ссылки на этот компонент используется весьма тривиальная разметка:**

```
<a href="{concat('#', @component-id)}">
  <xsl:value-of select="name"/>
</a>
```

Фрагмент генерирует разметку HTML `<a href="#C28392-33-TT">Turnip Twaddler</a>` для любой ссылки на компонент `Turnip Twaddler`. Если в вашем документе используются атрибуты `ID`, `IDREF` и `IDREFS`, построить необходимые ссылки будет несложно.

На рис. 6.1 показано, как выглядит документ HTML.

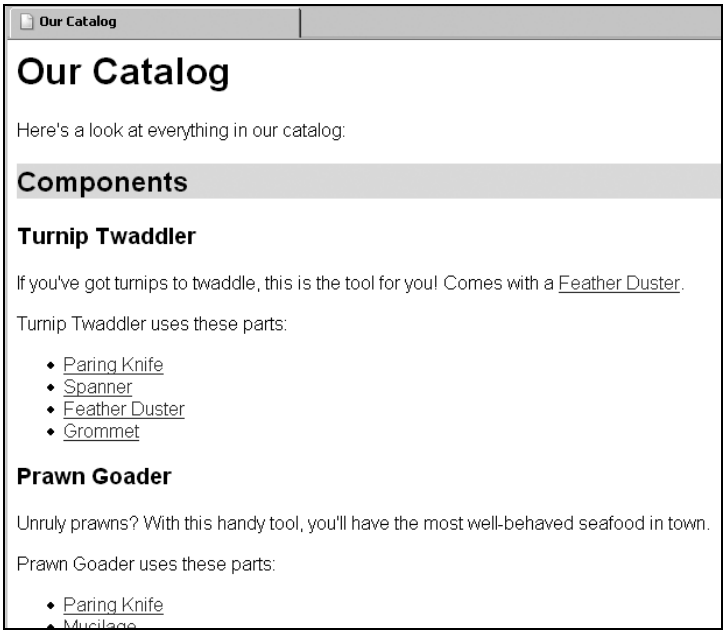


***Рис. 6.1.** Файл HTML со сгенерированными гиперссылками*

Ограничения типа данных ID

Пока что нам удавалось легко генерировать необходимые перекрестные ссылки. Однако тип данных `ID` и функция `id()` обладают рядом ограничений:

- Чтобы использовать тип `ID`, необходимо объявить атрибуты, использующие этот тип данных, в DTD или схеме. К сожалению, если описание DTD определяется как внешнее по отношению к документу XML, парсер XML не обязан читать его. А если описание DTD не прочитано, парсер понятия не имеет, относится ли заданный атрибут к типу `ID`. Аналогично при использовании схемы вам придется позаботиться о том, чтобы процессор XSLT проверил действительность документа XML по схеме и убедился в правильности использования типов данных `ID`, `IDREF` и `IDREFS`.
- Отношения между `ID`, `IDREF` и `IDREFS` должны определяться в документе XML. С другой стороны, было бы удобнее, чтобы в документе

XML определялись только данные, а отношения между частями документа определялись во внешнем источнике (например, в таблице стилей). Если вам потребуется определить новое отношение между частями документа, вы создаете новую таблицу стилей, а документ XML останется неизменным. Решение, при котором вам приходится изменять структуру документа XML каждый раз, когда потребуется определить новое отношение между частями документа, получается слишком громоздким и неудобным.

- Элемент может иметь не более одного атрибута типа `ID`. Если потребуется сослаться на один элемент несколькими способами, использовать функцию `id()` не удастся.
- Любое значение `ID` может присутствовать не более чем в одном элементе. Если вам потребуется сослаться на несколько элементов по одному значению, использовать функцию `id()` для этой цели также не удастся.
- Для всего документа существует только один набор `ID`. Другими словами, если вы объявите атрибуты `component-id` и `part-id` типа `ID`, значение `component-id` должно быть уникальным для всех атрибутов типа `ID`. Значение `component-id` не может совпадать со значением `part-id`, даже если эти атрибуты принадлежат разным элементам.
- При использовании DTD `ID` может быть только атрибутом элемента XML. Возможен только один способ использования функции `id()` для ссылки на другой элемент – через его атрибут типа `ID`. Если вы захотите найти другой элемент по атрибуту, не относящемуся к типу `ID`, или на основании содержимого этого элемента, или по его дочерним элементам и т. д., функция `id()` вам не поможет.
- XML Schema позволяет определить элемент с типом данных `xs:ID`; это означает, что для поиска элемента может использоваться функция `id()`. Данная возможность немного улучшает ситуацию, но требует схемосовместимого парсера XSLT.
- Значение `ID` должно быть именем XML. Другими словами, оно не может содержать пробелы, не может начинаться с цифры, и на него распространяются все остальные ограничения имен XML. (Эти ограничения определяются в разделе 2.3 XML Recommendation; за дополнительной информацией обращайтесь по адресу *http://www.w3.org/TR/REC-xml*.)

Для обхода этих ограничений в XSLT определяются элемент `<xsl:key>` и функция `key()`. Они рассматриваются в следующем разделе.

Функция key() и элемент <xsl:key>

После весьма подробного описания функции `id()` можно перейти к функции XSLT `key()` и элементу `<xsl:key>`. Элемент `<xsl:key>` определяет ключ, т. е. фактически строит индекс для документа. Построен-

ный индекс может использоваться для поиска всех элементов, обладающих определенным свойством. После определения ключа выборка нужных частей документа осуществляется функцией `key()`.

Допустим, имеется база данных почтовых адресов, которая индексируется по фамилиям адресатов, по штатам, почтовым индексам и т. д. На построение индексов тратится некоторое время, но позднее эти затраты компенсируются временем на обработку. (Учтите, что индексирование также может быть сопряжено со значительными затратами памяти, особенно в очень больших документах.) Если вам потребуется найти всех людей, живущих в штате Айдахо, записи можно будет отобрать напрямую по индексу; поиск по всей базе данных будет излишним.

В этом разделе мы подробно рассмотрим, как работают ключи, и сравним их возможности с функцией `id()`.

Определение ключа элементом <xsl:key>

Ключ определяется при помощи элемента `<xsl:key>`:

```
<xsl:key name="supplier-by-country" match="supplier" use="@country"/>
<xsl:key name="part-by-supplier" match="part" use="@supplier"/>
```

Ключ обладает тремя атрибутами:

`name`

Атрибут используется для ссылок на конкретный ключ. Если вам потребуется найти некоторую часть документа XML, нужный ключ задается как значение атрибута `name`.

`match`

Атрибут содержит выражение XPath, определяющее индексируемую часть документа. В приведенном выше примере создаются два ключа: для выборки элементов `<supplier>` и для выборки элементов `<part>`.

`use`

Атрибут содержит еще одно выражение XPath, интерпретируемое в контексте атрибута `match`. Иначе говоря, первый из приведенных ранее элементов `<xsl:key>` (с именем `supplier-by-country`) создает индекс для всех элементов `<supplier>`, используя атрибут `country` для их выборки. Второй элемент `<xsl:key>` (с именем `part-by-supplier`) создает индекс всех элементов `<part>`, используя для поиска атрибут `supplier`.

[2.0] В XSLT 2.0 появился четвертый атрибут `collation`, который позволяет задать набор правил сравнения значений. Пример, часто встречающийся в спецификациях: в немецком языке слово *улица* может записываться в двух вариантах – *Strasse* и *Straße*. При использовании немецкой функции упорядочения в определении ключа эти два слова будут считаться одинаковыми, хотя две строки очевидным образом различаются.

В спецификации XSLT 1.0 явно указано, что атрибуты match и use не могут содержать переменных.

Построение ссылок функцией key()

В измененном документе со списком деталей, который мы рассматривали совсем недавно, в каждый элемент <part> был добавлен элемент <supplier>, а в элемент <supplier> был добавлен атрибут country. Взглянув на структуру документа, определяемую во встроенном описании DTD (или во внешней схеме), вы заметите, что атрибуты vendor-id и country элемента <supplier> не относятся к типу данных ID, а атрибут supplier элемента <part> не относится к типу IDREF.

Мы хотим получить все детали, поставляемые из определенной страны. Если определить атрибут country с типом ID, для каждой страны можно будет создать только одного поставщика. Разумеется, для нашего документа такое ограничение неприемлемо.

Для обработки нового, более гибкого документа XML будет использоваться функция key(). В нашем примере определяются два ключа. Первый ключ возвращает все элементы <supplier> для заданного названия страны. Второй ключ возвращает все элементы <part>, у которых атрибут <supplier> содержит заданный идентификатор поставщика. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- key.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:param name="country-name"/>

  <xsl:key name="supplier-by-country" match="supplier" use="@country"/>
  <xsl:key name="part-by-supplier" match="part" use="@supplier"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>
          <xsl:text>Parts from </xsl:text>
          <xsl:value-of select="$country-name"/>
        </title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>
          <xsl:text>Parts from </xsl:text>
          <xsl:value-of select="$country-name"/>
        </h1>
        <xsl:choose>
          <xsl:when test="key('supplier-by-country', $country-name)">
```

```
            <xsl:apply-templates select="key('supplier-by-country',
                                             $country-name)"/>
          </xsl:when>

          <xsl:otherwise>
            <p>Sorry, we don't get any parts from that country!</p>
          </xsl:otherwise>
        </xsl:choose>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="supplier">
    <h2>
      <xsl:value-of select="name"/>
    </h2>
    <p>
      <xsl:value-of select="name"/>
      <xsl:text> supplies these parts:</xsl:text>
    </p>
    <ul>
      <xsl:for-each select="key('part-by-supplier', @vendor-id)">
        <li>
          <b>
            <xsl:value-of select="name"/>
          </b>
          <xsl:text>:  </xsl:text>
          <xsl:apply-templates select="description"/>
        </li>
      </xsl:for-each>
    </ul>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей получает внешний параметр `country-name` **(определяемый в начале таблицы стилей) и возвращает все детали, поставляемые компаниями из указанной страны. При первом использовании функции** `key()` **мы проверяем, имеются ли значения, соответствующие внешнему параметру:**

```
<xsl:when test="key('supplier-by-country', $country-name)">
  <xsl:for-each select="key('supplier-by-country', $country-name)">
    <xsl:apply-templates select="."/>
  </xsl:for-each>
</xsl:when>
```

Ключ `country-index` **возвращает все элементы** `<supplier>`**, соответствующие заданному значению** `$country-name`**. Если будет получен хотя бы один элемент, мы обрабатываем его (а если нет, выражение дает результат** `false`**). При обработке элементов** `<supplier>` **используется другой ключ:**

```
<xsl:for-each select="key('part-by-supplier', @vendor-id)">
```

Конструкция возвращает все элементы <part>, у которых атрибут supplier совпадает с атрибутом vendor-id текущего элемента <supplier>.

Обратите внимание: атрибут, который мы будем использовать для выборки совпадающих узлов, может содержать пробелы. Если бы атрибут country относился к типу данных ID, он бы не мог содержать значение "Great Britain". В этом проявляется одно из преимуществ ключей. Документ HTML, создаваемый для параметра country-name со значением "Great Britain", показан на рис. 6.2.

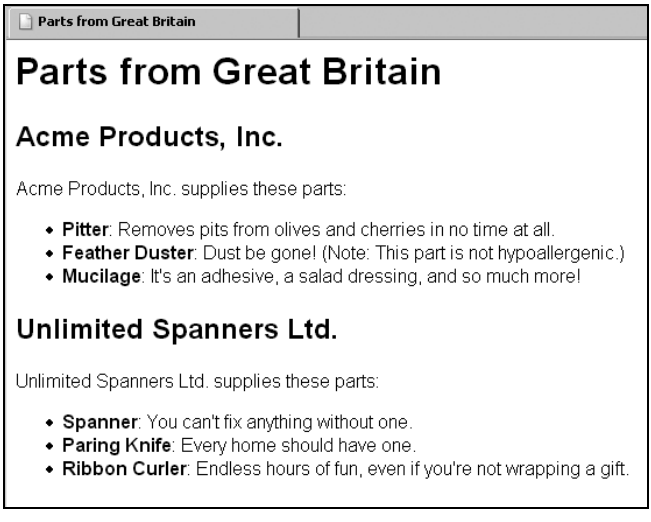


Рис. 6.2. *Все детали от поставщиков из Великобритании*

Преимущества функции key()

Присмотревшись поближе к функции key(), мы видим, что она обладает рядом преимуществ:

- Функция key() определяется в таблице стилей. Это означает, что вы можете в любой момент определить сколько угодно отношений между частями документа XML. Если завтра вдруг потребуется определить новое отношение, вам не нужно будет изменять документ XML.
- Для каждого элемента можно определить сколько угодно функций key(). В нашем примере со списками деталей можно определить функции key() для значений vendor-id, part-id и component-id. Мы также можем создать функции key(), зависящие от текста различных элементов или их потомков. Если бы вместо функции key() использовались типы данных ID, выбор ограничивался бы одним индексом, основанным на значении одного атрибута типа данных ID.

Подведем частичный итог: для элемента можно определить более одного ключа `key()`, и этот ключ не обязан базироваться на атрибуте. Ключ может базироваться на тексте элемента, тексте дочерних элементов или других конструкциях.

- Одному значению может соответствовать любое количество элементов. Еще раз вернемся к нашему примеру: когда мы используем функцию `key()` для получения всех деталей от поставщиков из конкретной страны, функция `key()` возвращает набор узлов произвольного размера. Если бы мы использовали `ID`, каждой стране мог бы соответствовать только один элемент.
- Значения, используемые для поиска элементов по ключу, не ограничиваются именами XML. При использовании типа данных `ID` значение не может содержать пробелы, а также подчиняется другим ограничениям.

Обычно используется версия функции `key()` с двумя аргументами. Она получает имя ключа и искомое значение и возвращает совпавшие объекты. Также имеется версия функции с тремя аргументами, которая позволяет ограничить значения, возвращаемые функцией, конкретным набором узлов. За дополнительной информацией обращайтесь к описанию функции `key()` в приложении C.

Генерирование ссылок в неструктурированных документах

Прежде чем окончательно расстаться с темой ссылок, рассмотрим еще один полезный прием. До настоящего момента все примеры этой главы были четко структурированы. Если между двумя информационными объектами существовало отношение, мы создавали связь между ними при помощи пары `ID/IDREF`. А если в преобразуемом документе XML такой структуры нет? К счастью, ее можно создать при помощи функций `key()` и `generate-id()`.

Неструктурированный документ XML для создания ссылок

В следующем примере из документа XML были удалены все атрибуты `id` и `refid`, которые нам так пригодились прежде. Пример получился искусственным, но он показывает, как использовать функции `key()` и `generate-id()` для создания ссылок между частями документа.

Документ XML, из которого исключены все ссылки, связывавшие прежде разные части документа:

```
<?xml version="1.0"?>
<!-- parts-list4.xml -->
<parts-list>
  <component>
```

```
    <name>Turnip Twaddler</name>
    <partref>Paring Knife</partref>
    <partref>Spanner</partref>
    <partref>Feather Duster</partref>
    <partref>Grommet</partref>
    <description>
      If you've got turnips to twaddle, this is the tool for you!
      Comes with a <partref>Feather Duster</partref>.
    </description>
  </component>

...
  <part>
    <name>Pitter</name>
    <description>
      Removes pits from olives and cherries in no time at all.
    </description>
  </part>
  <part>
    <name>Patter</name>
    <description>
      We're not sure what these things do, but people seem
      to like 'em.
    </description>
  </part>
...
</parts-list>
```

Из документа удалены все атрибуты `ID` и `IDREF`. Для элементов, ранее использовавших атрибуты для связывания разных частей документа (например, `<partref>`), мы воспользуемся текстом элемента, на который создается ссылка. Для построения системы перекрестных ссылок, наподобие созданной ранее, необходимо сделать три вещи:

1. Определить два ключа для всех частей и компонентов. Один ключ возвращает элемент `<part>` с заданным именем, а другой находит элемент `<component>` с дочерним элементом `<partref>`, текст которого соответствует заданному имени.
2. Сгенерировать новый идентификатор для каждого обнаруженного элемента `<component>` и `<part>`.
3. Для каждого элемента `<component>` использовать один ключ для выборки узлов `<part>`, соответствующих некоторому имени. Для каждого элемента `<part>` второй ключ используется для выборки узлов `<component>`, содержащих ссылку на текущую деталь. Идентификаторы генерируются функцией `generate-id()`.

А теперь рассмотрим соответствующие части таблицы стилей. Сначала в ней определяются два ключа:

```
<xsl:key name="parts-index" match="part/name" use="."/>
<xsl:key name="component-index" match="component" use="partref"/>
```

Первый ключ возвращает элемент <name> для заданного названия детали. Обратите внимание: атрибут match означает, что мы получаем элемент <name>; если бы нас интересовал сам элемент <part>, мы бы воспользовались родительской осью узла, возвращаемого функцией key(). (В этой таблице стилей нам не нужно обращаться к элементу <part>; это объясняет, почему ключ создается именно таким образом.)

Второй ключ возвращает элемент <component>, у которого элемент <partref> совпадает с заданным названием детали. От функции key() мы получаем <component> – родителя как элемента <partref>, содержащего искомое название детали, так и элемента <name>, вставляемого в документ HTML.

На следующем этапе создаются идентификаторы для всех элементов <component> и <part>:

```
<xsl:template match="component">
  <a name="{generate-id(name)}"/>
...
</xsl:template>
...
<xsl:template match="part">
  <a name="{generate-id(name)}"/>
...
</xsl:template>
```

В обоих случаях идентификатор генерируется на основании текста дочернего элемента <name> заданного элемента. Имена деталей и компонентов используются в таблице стилей, так что генерирование идентификаторов на базе элементов <name> упрощает задачу.

Теперь необходимо обработать все элементы <partref> для заданного элемента <component>. Вот как это делается:

```
<ul>
  <xsl:for-each select="partref">
    <li>
      <a href="{concat('#', generate-id(key('parts-index', .)[1]))}">
        <xsl:value-of select="."/>
      </a>
    </li>
  </xsl:for-each>
</ul>
```

Атрибут href гиперссылки генерируется по идентификатору первого совпадения функции key(). Ключ parts-index возвращает элемент <name>, соответствующий строке; в данном случае строка является текстовым значением текущего элемента.

Обратите внимание на использование предикатного выражения [1] для определения первого объекта набора узлов или последовательности. В XSLT 1.0 это просто желательно, поскольку такая запись четко указывает, какой узел нас интересует. Однако в XSLT 2.0 это абсолют-

но необходимо, потому что передача последовательности с несколькими узлами функции `generate_id()` в таблицах стилей XSLT 2.0 приводит к фатальной ошибке.

Осталось решить последнюю задачу: создать ссылки от каждого элемента `<part>` ко всем элементам `<component>`, в которых он используется. Код выглядит так:

```
<ul>
  <xsl:for-each select="key('component-index', name)">
    <li>
      <a href="{concat('#', generate-id(name))}">
        <xsl:value-of select="name"/>
      </a>
    </li>
  </xsl:for-each>
</ul>
```

Вспомните, что ключ `component-index` возвращает элемент `<component>`, являющийся родителем как элемента `<partref>`, так и элемента `<name>`. В качестве критерия поиска ключу передается название текущей детали. Получив от ключа элемент `<component>`, мы используем его дочерний элемент `<name>` для генерирования идентификатора и вывода названия компонента.

Полная таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- generate-id.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:key name="parts-index" match="part/name" use="."/>
  <xsl:key name="component-index" match="component" use="partref"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Our Catalog</title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>Our Catalog</h1>
        <p>Here's a look at everything in our catalog:</p>
        <h2 style="background: #66FF66;">Components</h2>
        <xsl:apply-templates select="/parts-list/component"/>
        <h2 style="background: #6666FF;">Parts</h2>
        <xsl:apply-templates select="/parts-list/part"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="component">
```

```
  <a name="{generate-id(name)}"/>
  <h3>
    <xsl:value-of select="name"/>
  </h3>
  <p>
    <xsl:apply-templates select="description"/>
  </p>
  <p>
    <xsl:value-of select="name"/>
    <xsl:text> uses these parts:</xsl:text>
  </p>
  <ul>
    <xsl:for-each select="partref">
      <li>
        <a href="{concat('#', generate-id(key('parts-index', .)[1]))}">
          <xsl:value-of select="."/>
        </a>
      </li>
    </xsl:for-each>
  </ul>
</xsl:template>

<xsl:template match="description">
  <xsl:apply-templates select="*|text()"/>
</xsl:template>

<xsl:template match="component/description/partref">
  <a href="{concat('#', generate-id(key('parts-index', .)[1]))}">
    <xsl:value-of select="."/>
  </a>
</xsl:template>

<xsl:template match="part">
  <a name="{generate-id(name)}"/>
  <h3>
    <xsl:value-of select="name"/>
  </h3>
  <p>
    <xsl:apply-templates select="description"/>
  </p>
  <p>
    <xsl:value-of select="name"/>
    <xsl:text> is used in these components:</xsl:text>
  </p>
  <ul>
    <xsl:for-each select="key('component-index', name)">
      <li>
        <a href="{concat('#', generate-id(name))}">
          <xsl:value-of select="name"/>
        </a>
      </li>
    </xsl:for-each>
```

```
    </ul>
  </xsl:template>

</xsl:stylesheet>
```

Если просмотреть вывод HTML в броузере, документ выглядит точно так же, как с более ранней таблицей стилей. Конечно, исходный код HTML немного изменился:

```
<a name="d1e31"></a>
<h3>Prawn Goader</h3>
<p>
  Unruly prawns?  With this handy tool, you'll have the most
  well-behaved seafood in town.
</p>
<p>Prawn Goader uses these parts:</p>
<ul>
  <li><a href="#d1e181">Paring Knife</a></li>
  <li><a href="#d1e190">Mucilage</a></li>
  <li><a href="#d1e199">Ribbon Curler</a></li>
</ul>
```

Все имена точек связывания были сгенерированы процессором XSLT. Вероятно, другой процессор XSLT сгенерирует для этих идентификаторов другие значения, но они тоже будут работать. Более того, даже при повторной обработке документа и таблицы стилей тем же процессором могут быть сгенерированы другие значения. (Функция `generate-id()` подробно описана в следующем разделе.)

Использование функций `key()` и `generate-id()` позволило автоматически создать идентификаторы и ссылки. Такое решение не идеально; нам приходится проверять, что текст элемента `<partref>` точно соответствует тексту элемента `<name>` детали. Несмотря на это, функция `generate-id()` помогает создать в документе некоторую упорядоченную структуру.

Функция generate-id()

Прежде чем переходить к следующей теме, стоит поподробнее рассказать о функции `generate-id()`. Функция получает аргумент с набором узлов и работает следующим образом:

- В рамках конкретного преобразования каждый раз, когда `generate-id()` вызывается для заданного узла, она возвращает один и тот же идентификатор. Если запустить преобразование на следующий день, нет гарантий, что `generate-id()` выдаст тот же идентификатор, что и сегодня. Но во время работы процессора XSLT при многократном вызове `generate-id()` для конкретного узла всегда будут возвращаться одинаковые идентификаторы.
- При вызове `generate-id()` для двух разных узлов будут сгенерированы два разных идентификатора.

- *[1.0]* Для набора узлов `generate-id()` возвращает идентификатор узла, следующего первым в порядке документа.

  [2.0] В XSLT 2.0 передача `generate-id()` последовательности, содержащей более одного объекта, приводит к фатальной ошибке.
- Если функции передается пустой набор узлов (скажем, при вызове `generate-id(fleeber)`, когда в текущем контексте нет ни одного элемента `<fleeber>`), `generate-id()` возвращает пустую строку.
- Если набор узлов не указан (вызов вида `generate-id()`), функция генерирует идентификатор для контекстного узла.

Функция `generate-id()` не обязана проверять, не совпадает ли сгенерированный идентификатор с другим идентификатором, уже присутствующим в документе. Другими словами, если документ содержит атрибут `ID` со значением `sdk3829a`, не исключено, что генерируемое `generate-id()` значение окажется равным `sdk3829a`. Это маловероятно, но теоретически возможно.

Итоги

В этой главе мы рассмотрели несколько способов создания связей и построения перекрестных ссылок между разными частями документа. Если документ XML изначально структурирован, вы можете воспользоваться функциями `id()` и `key()` для определения разнообразных связей между частями документа. Даже если документ XML не структурирован, функции `key()` и `generate-id()` помогут создать простые ссылки. Следующая глава посвящена сортировке и группировке – двум другим способам упорядочения информации в документах XML.

7

Сортировка и группировка элементов

Надеюсь, мне удалось достаточно убедительно показать, что XSLT позволяет преобразовывать большие наборы данных XML в другие полезные форматы. Во всех рассмотренных примерах исходный документ XML обрабатывался в так называемом *порядке документа*. Однако нам хотелось бы иметь возможность обрабатывать документы XML другими стандартными способами:

- Сортировать элементы XML (частично или полностью), а затем генерировать результат на основании отсортированных элементов.
- Группировать данные, выделив все элементы по некоторому общему свойству, а затем отсортировывать группы элементов.

Примеры таких операций будут рассмотрены в этой главе.

Сортировка данных при помощи элемента <xsl:sort>

Данные в формате XML проще всего отсортировать при помощи элемента `<xsl:sort>`. Этот элемент временно изменяет упорядочение коллекции элементов на основании критерия, определяемого в таблице стилей.

Первый пример

В первом примере используется документ с почтовыми адресами на территории США. (Не сочтите за шовинизм; естественно, в каждой стране существуют свои правила оформления почтовых адресов. Нам просто нужен короткий пример документа, который можно отсортировать несколькими полезными способами.) Исходный документ выглядит так:

```
<?xml version="1.0"?>
<!-- names.xml -->
<addressbook>
  <address>
    <name>
      <title>Mr.</title>
      <first-name>Chester Hasbrouck</first-name>
      <last-name>Frisby</last-name>
    </name>
    <street>1234 Main Street</street>
    <city>Sheboygan</city>
    <state>WI</state>
    <zip>48392</zip>
  </address>
  <address>
    <name>
      <first-name>Mary</first-name>
      <last-name>Backstayge</last-name>
    </name>
    <street>283 First Avenue</street>
    <city>Skunk Haven</city>
    <state>MA</state>
    <zip>02718</zip>
  </address>
  <address>
    <name>
      <title>Ms.</title>
      <first-name>Natalie</first-name>
      <last-name>Attired</last-name>
    </name>
    <street>707 Breitling Way</street>
    <city>Winter Harbor</city>
    <state>ME</state>
    <zip>00218</zip>
  </address>
  <address>
    <name>
      <first-name>Harry</first-name>
      <last-name>Backstayge</last-name>
    </name>
    <street>283 First Avenue</street>
    <city>Skunk Haven</city>
    <state>MA</state>
    <zip>02718</zip>
  </address>
  <address>
    <name>
      <first-name>Mary</first-name>
      <last-name>McGoon</last-name>
    </name>
    <street>103 Bryant Street</street>
```

```
    <city>Boylston</city>
    <state>VA</state>
    <zip>27318</zip>
  </address>
  <address>
    <name>
      <title>Ms.</title>
      <first-name>Amanda</first-name>
      <last-name>Reckonwith</last-name>
    </name>
    <street>930-A Chestnut Street</street>
    <city>Lynn</city>
    <state>MA</state>
    <zip>02930</zip>
  </address>
</addressbook>
```

Требуется сгенерировать список этих адресов, упорядоченных по значению `<last-name>`**. Для выполнения сортировки мы воспользуемся волшебным элементом** `<xsl:sort>`**. Таблица стилей выглядит так:**

```
<?xml version="1.0"?>
<!-- namesorter1.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:for-each select="addressbook/address">
      <xsl:sort select="name/last-name"/>
      <xsl:if test="name/title">
        <xsl:value-of select="name/title"/>
        <xsl:text> </xsl:text>
      </xsl:if>
      <xsl:value-of select="name/first-name"/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="name/last-name"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:value-of select="street"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:value-of select="city"/>
      <xsl:text>, </xsl:text>
      <xsl:value-of select="state"/>
      <xsl:text>  </xsl:text>
      <xsl:value-of select="zip"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Ключевая роль в этой таблице стилей отводится элементам <xsl:for-each> и <xsl: sort>. Элемент <xsl:for-each> выбирает объекты для обработки, а элемент <xsl:sort> упорядочивает их перед выводом. (Стоит также упомянуть элемент <xsl:if>, проверяющий наличие у клиента титульной приставки.)

Обратите внимание на элемент <xsl:output method="text" />, генерирующий текстовый файл. (При желании вы можете сгенерировать файл HTML или что-нибудь более сложное.) Первая попытка сортировки дает следующий результат:

```
Ms. Natalie Attired
707 Breitling Way
Winter Harbor, ME  00218

Mary Backstayge
283 First Avenue
Skunk Haven, MA  02718

Harry Backstayge
283 First Avenue
Skunk Haven, MA  02718

Mr. Chester Hasbrouck Frisby
1234 Main Street
Sheboygan, WI  48392

Mary McGoon
103 Bryant Street
Boylston, VA  27318

Ms. Amanda Reckonwith
930-A Chestnut Street
Lynn, MA  02930
```

Как видно из выходных данных, адреса из исходного документа были отсортированы по фамилиям. Все, что для этого потребовалось, – включить <xsl:sort> в таблицу стилей, и порядок следования элементов изменился словно по волшебству. Если вы еще сомневаетесь в том, что XSLT может повысить продуктивность вашей работы, попробуйте написать эквивалентный код Java с вызовами методов DOM.

Таблицу стилей можно усовершенствовать сортировкой по элементу <first-name> внутри <last-name>. В нашем примере запись Mary Backstayge должна следовать после Harry Backstayge. Таблица стилей с сортировкой по нескольким ключам выглядит так:

```
<?xml version="1.0"?>
<!-- namesorter2.xsl -->
...
<xsl:template match="/">
  <xsl:for-each select="addressbook/address">
    <xsl:sort select="name/last-name"/>
    <xsl:sort select="name/first-name"/>
    ...
```

Мы просто добавляем в таблицу стилей второй элемент <xsl:sort>. Он делает именно то, что требуется: сортирует элементы <address> по значениям <first-name> при совпадении <last-name>.

Теперь результат выглядит лучше:

```
Ms. Natalie Attired
707 Breitling Way
Winter Harbor, ME  00218

Harry Backstayge
283 First Avenue
Skunk Haven, MA  02718

Mary Backstayge
283 First Avenue
Skunk Haven, MA  02718

Mr. Chester Hasbrouck Frisby
1234 Main Street
Sheboygan, WI  48392

Mary McGoon
103 Bryant Street
Boylston, VA  27318

Ms. Amanda Reckonwith
930-A Chestnut Street
Lynn, MA  02930
```

Подробнее об элементе <xsl:sort>

Итак, мы рассмотрели пару примеров того, как работает элемент <xsl:sort>; теперь можно переходить к рассмотрению его синтаксиса, атрибутов и области применения.

А что это за синтаксис?

Хорошо, что вы спросили. Рабочая группа XSLT могла бы реализовать его примерно так:

```
<xsl:for-each select="addressbook/address" sort-key-1="name/last-name"
  sort-key-2="name/first-name"/>
```

У такого подхода есть один недостаток: сколько бы атрибутов sort-key-x вы ни определили, кто-нибудь просто из вредности заявит, что ему на самом деле нужен атрибут sort-key-8293. Чтобы обойти этот неприятный момент, проектировщики XSLT выбрали возможность задания ключей сортировки в нескольких элементах <xsl:sort>. Первый элемент определяет первичный ключ сортировки, второй – вторичный ключ, 8293-й – восемь-тысяч-двести-девяносто-третичный и т. д.

Это объясняет, почему синтаксис выглядит именно так, а не иначе. Но как он работает? Когда я впервые увидел запись вида:

```
<xsl:for-each select="addressbook/address">
  <xsl:sort select="name/last-name"/>
  <xsl:sort select="name/first-name"/>
  ...
</xsl:for-each>
```

я решил, что все узлы сортируются при каждой итерации элемента <for-each>. Такое решение было бы чудовищно неэффективным; если все узлы уже отсортированы, зачем сортировать их заново? В действительности процессор XSLT начинает с обработки всех элементов <xsl:sort>, а затем обрабатывает элемент <xsl:for-each> так, словно элементы <xsl:sort> не включены в цикл.

Следующая запись менее эффективна, но если она кажется вам более разумной с точки зрения синтаксиса, вы можете создать и такую таблицу стилей:

```
<xsl:template match="/">
  <xsl:for-each select="addressbook/address">
    <xsl:sort select="name/last-name"/>
    <xsl:sort select="name/first-name"/>
    <xsl:for-each select="."> <!-- Медленно, но работает. -->
      <xsl:apply-templates/>
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>
```

(Не надо так делать, показываю только в учебных целях.) Эта таблица стилей выдает те же результаты, что и предыдущая.

Другой способ основан на использовании элемента <xsl:sort> в элементе <xsl:apply-templates>. Соответствующая таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- namesorter3.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:apply-templates select="addressbook/address">
      <xsl:sort select="name/last-name"/>
      <xsl:sort select="name/first-name"/>
    </xsl:apply-templates>
  </xsl:template>

  <xsl:template match="address">
    <xsl:if test="name/title">
      <xsl:value-of select="name/title"/>
      <xsl:text> </xsl:text>
    </xsl:if>
    <xsl:value-of select="name/first-name"/>
    <xsl:text> </xsl:text>
```

```
      <xsl:value-of select="name/last-name"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:value-of select="street"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:value-of select="city"/>
      <xsl:text>, </xsl:text>
      <xsl:value-of select="state"/>
      <xsl:text>  </xsl:text>
      <xsl:value-of select="zip"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:text>&#xA;</xsl:text>
    </xsl:template>

  </xsl:stylesheet>
```

Таблица стилей, в которой <xsl:sort> используется внутри <xsl:apply-templates>, выдает те же результаты, что и предыдущая.

Атрибуты

Элемент <xsl:sort> имеет несколько атрибутов; самые полезные атрибуты описаны в этом разделе. Полная информация обо всех атрибутах <xsl:sort> приведена в приложении А.

select

Атрибут select определяет характеристики сортировки. Его содержимое представляет собой выражение XPath и может использоваться для выбора элементов, текста, атрибутов, комментариев, узлов-«предков» и т. д. Как всегда, выражение XPath, определяемое в select, обрабатывается в контексте содержащего его элемента. Пример:

```
<xsl:template match="/">
  <xsl:apply-templates select="addressbook/address">
    <xsl:sort select="name/last-name"/>
    <xsl:sort select="name/first-name"/>
  </xsl:apply-templates>
</xsl:template>
```

В этом примере атрибуты select элементов <xsl:sort> интерпретируются в контексте выражения addressbook/address. Это означает, что конструкция select="name/last-name" задает элемент <last-name> из элемента <name> в элементе <address>, находящемся внутри элемента <addressbook>.

data-type

Атрибут data-type имеет три допустимых значения:

- data-type="text"
- data-type="number"
- data-type="QName" с определением конкретного типа данных. Способ поддержки этого типа данных (и даже само ее наличие) определяется реализацией.

В спецификации XSLT определяется поведение для значений data-type="text" и data-type="number". Рассмотрим следующий документ XML:

```
<?xml version="1.0"?>
<!-- numberlist.xml -->
<numberlist>
  <number>127</number>
  <number>23</number>
  <number>10</number>
</numberlist>
```

Мы отсортируем эти значения с использованием стандартного типа данных text (также можно явно указать атрибут data-type="text" для получения тех же результатов):

```
<?xml version="1.0"?>
<!-- sort-datatype-text.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:for-each select="numberlist/number">
      <xsl:sort select="."/>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Если значения элементов интерпретируются как текст, мы получаем следующий результат:

```
10
127
23
```

Дело в том, что при текстовой сортировке все данные, начинающиеся с "1", предшествуют всем данным, начинающимся с "2". Приведем элемент <xsl:sort> к виду <xsl:sort select="." data-type="number"/>:

```
<?xml version="1.0"?>
<!-- sort-datatype-number1.xsl -->
...
  <xsl:template match="/">
    <xsl:for-each select="numberlist/number">
      <xsl:sort select="." data-type="number"/>
      ...
```

На этот раз результат будет выглядеть так:

```
10
27
123
```

Если указать другой тип данных (например, `data-type="floating-point"`), невозможно заранее предсказать, что сделает процессор XSLT. Спецификация XSLT позволяет использовать другие значения, но процессор XSLT сам решает, как они должны обрабатываться (и должны ли обрабатываться вообще). За информацией о поддержке других значений, кроме `data-type="text"` и `data-type="number"`, обращайтесь к документации процессора.

Последнее замечание: если в элементе указан атрибут `data-type="number"`, а значения не являются числовыми, то при сортировке все нечисловые значения будут предшествовать всем числовым значениям. Таким образом, с атрибутом `order="ascending"` нечисловые значения будут сгруппированы в начале, а с атрибутом `order="descending"` они группируются в конце.

```
<?xml version="1.0"?>
<!-- badnumberlist.xml -->
<numberlist>
  <number>127</number>
  <number>23</number>
  <number>zzz</number>
  <number>10</number>
  <number>yyy</number>
</numberlist>
```

Для этих разнородных данных правильный результат сортировки будет выглядеть так:

```
zzz
yyy
10
23
127
```

Обратите внимание: нечисловые значения не сортируются; они просто помещаются в выходной документ в том порядке, в котором они встретились при обработке исходного документа.

[2.0] Атрибут `data-type` считается устаревшим в XSLT 2.0. Рекомендуемый способ сортировки типизованных данных в XSLT 2.0 основан на задании типа в атрибуте `select` самого элемента `<xsl:sort>`. Например, элемент `<xsl:sort select="xs:integer(.)"/>` интерпретирует все сортируемые объекты как целые числа:

```
<?xml version="1.0"?>
<!-- sort-datatype-number2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
```

```
    <xsl:for-each select="numberlist/number">
      <xsl:sort select="xs:integer(.)"/>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Учтите, что при использовании таблицы стилей XSLT 2.0 с файлом *badnumberlist.xml* произойдет ошибка времени выполнения. Вызов `xs:integer('zzz')` не поможет.

`order`

Порядок сортировки: по возрастанию (`order="ascending"`) или по убыванию (`order="descending"`). По умолчанию используется значение `order="ascending"`.

`case-order`

Атрибут может иметь два значения. `case-order="upper-first"` означает, что прописные буквы должны предшествовать строчным, а в режиме `case-order="lower-first"` строчные буквы находятся на первом месте. Атрибут `case-order` используется только в том случае, если атрибут `data-type` содержит значение `text`. Значение по умолчанию зависит от значения описываемого ниже атрибута `lang`.

`lang`

Атрибут определяет язык ключей сортировки. Допустимые значения атрибута совпадают с допустимыми значениями атрибута `xml:lang`, определяемыми в разделе 2.12 спецификации XML 1.0. Коды языков часто используются в программировании Java, локальных контекстах UNIX и других местах, в которых определяются языки и страны по стандарту ISO. Например, `lang="en"` соответствует английскому языку, `lang="en-US"` – американскому английскому, а `lang="en-GB"` – британскому английскому. Без атрибута `lang` (а на практике этот атрибут применяется редко) процессор XSLT определяет язык по умолчанию из системной среды.

Где использовать <xsl:sort>?

Элемент `<xsl:sort>` может находиться в элементах `<xsl:apply-templates>` и `<xsl:for-each>`.

[2.0] В XSLT 2.0 элемент `<xsl:sort>` также может находиться в новых элементах `<xsl:for-each-group>` и `<xsl:perform-sort>`; подробности далее в этой главе.

Элементы `<xsl:sort>` (один или несколько) должны находиться на первом месте. Например, при попытке выполнения следующего фрагмента в процессоре XSLT произойдет исключение:

```
<xsl:for-each select="addressbook/address">
  <xsl:sort select="name/last-name"/>
```

```
  <xsl:value-of select="name/title"/>
  <xsl:sort select="name/first-name"/> <!-- НЕДОПУСТИМО! -->
```

Другой пример

Собственно, мы уже достаточно полно рассмотрели элемент <xsl:sort>. Давайте сделаем завершающий штрих и изменим таблицу стилей так, чтобы элемент xsl:sort работал с подмножеством адресов, а затем сортировал это подмножество. Сортировка будет применяться к адресам только тех штатов, чьи названия начинаются с буквы M. Как и следовало ожидать, задача решается при помощи выражения XPath, ограничивающего состав сортируемых элементов:

```
<?xml version="1.0"?>
<!-- namesorter4.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text" indent="no"/>

  <xsl:template match="/">
    <xsl:for-each select="addressbook/address[starts-with(state, 'M')]">
      <xsl:sort select="name/last-name"/>
      <xsl:sort select="name/first-name"/>
      <xsl:if test="name/title">
        <xsl:value-of select="name/title"/>
        <xsl:text> </xsl:text>
      </xsl:if>
      <xsl:value-of select="name/first-name"/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="name/last-name"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:value-of select="street"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:value-of select="city"/>
      <xsl:text>, </xsl:text>
      <xsl:value-of select="state"/>
      <xsl:text>  </xsl:text>
      <xsl:value-of select="zip"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

А вот и результат: адреса из штатов, начинающихся с буквы M, отсортированные сначала по фамилии, а затем по имени:

```
Ms. Natalie Attired
707 Breitling Way
Winter Harbor, ME  00218

Harry Backstayge
```

```
283 First Avenue
Skunk Haven, MA  02718

Mary Backstayge
283 First Avenue
Skunk Haven, MA  02718

Ms. Amanda Reckonwith
930-A Chestnut Street
Lynn, MA  02930
```

В элементе `xsl:for-each` выражение XPath содержит предикат, благодаря которому выбираются только адреса с элементом `<state>`, начинающимся с буквы `M`. Этот пример вплотную подводит нас к теме группировки узлов.

Со сгруппированными узлами можно также выполнить и другие операции, например:

- Сгенерировать список уникальных почтовых индексов с указанием количества адресов, использующих эти индексы.
- Для каждого уникального почтового индекса (или штата, или фамилии и т. д.) выполнить сортировку по соответствующему полю и вывести список всех адресов с указанным почтовым индексом.

Все эти возможности будут рассмотрены чуть позже. А прежде чем переходить к теме группировки, мы должны рассмотреть новый элемент `<xsl:perform-sort>`.

[2.0] Элемент <xsl:perform-sort>

Как упоминалось в главе 3, в XSLT 2.0 появилась концепция *последовательности* – группы узлов или атомарных значений. Последовательность обычно создается в ходе обработки таблицы стилей, чаще всего в переменной. Для сортировки последовательности можно воспользоваться элементом `<xsl:perform-sort>`. Все, что говорилось ранее о сортировке, в равной степени относится к `<xsl:perform-sort>`; рассмотрим несколько примеров.

Существуют два варианта использования элемента `<xsl:perform-sort>`: можно передать этому элементу существующую последовательность и отсортировать ее, а можно использовать элемент `<xsl:perform-sort>` и для создания, и для сортировки последовательности. В первом примере мы создадим последовательность из всех элементов `<city>` и используем `<xsl:perform-sort>` для ее сортировки:

```
<?xml version="1.0"?>
<!-- perform-sort1.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="sortedCities" as="xs:string*">
      <xsl:perform-sort select="addressbook/address/city">
        <xsl:sort select="."/>
      </xsl:perform-sort>
    </xsl:variable>
    <xsl:text>Our customers live in these cities:&#xA;&#xA;</xsl:text>
    <xsl:value-of select="$sortedCities" separator="&#xA;"/>
  </xsl:template>

</xsl:stylesheet>
```

Атрибут `select` элемента `<xsl:perform-sort>` определяет сортируемую последовательность. Результат применения этой таблицы стилей к адресной книге выглядит так:

```
Our customers live in these cities:

Boylston
Lynn
Sheboygan
Skunk Haven
Skunk Haven
Winter Harbor
```

Можно также воспользоваться новым элементом `<xsl:sequence>` для создания последовательности внутри `<xsl:perform-sort>`:

```
<?xml version="1.0"?>
<!-- perform-sort2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="sortedCities" as="xs:string*">
      <xsl:perform-sort>
        <xsl:sort select="."/>
        <xsl:sequence select="addressbook/address/city"/>
      </xsl:perform-sort>
    </xsl:variable>
    <xsl:text>Our customers live in these cities:&#xA;&#xA;</xsl:text>
    <xsl:value-of select="$sortedCities" separator="&#xA;"/>
  </xsl:template>

</xsl:stylesheet>
```

У такого решения нет сколько-нибудь ощутимых преимуществ, но эта таблица стилей выдает те же результаты. Вариант с размещением выражения XPath в `<xsl:perform-sort>` гораздо проще. Содержимое элемента `<xsl:perform-sort>` должно начинаться с одного или нескольких

элементов <xsl:sort>. Во втором примере атрибут select не используется, поэтому элемент <xsl:sequence> обеспечивает выборку данных для сортировки. Элемент <xsl:perform-sort> создает всю последовательность, а затем использует внутренние элементы <xsl:sort> для ее сортировки.

Чтобы получить данные из адресной базы клиентов по всему миру, мы можем добавить элементы <xsl:sequence> для выборки дополнительных данных:

```
<?xml version="1.0"?>
<!-- perform-sort3.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="sortedCities" as="xs:string*">
      <xsl:perform-sort>
        <xsl:sort select="."/>
        <xsl:apply-templates select="addressbook/address/city"/>
        <xsl:sequence select="('London', 'Adelaide', 'Rome')"/>
        <xsl:sequence select="('Jakarta', 'Sao Paulo', 'Timbuktu')"/>
      </xsl:perform-sort>
    </xsl:variable>
    <xsl:text>Our customers live in these cities:&#xA;&#xA;</xsl:text>
    <xsl:value-of select="$sortedCities" separator="&#xA;"/>
  </xsl:template>

</xsl:stylesheet>
```

Как и следовало ожидать, на этот раз результаты оказываются более впечатляющими:

```
Our customers live in these cities:

Adelaide
Boylston
Jakarta
London
Lynn
Rome
Sao Paulo
Sheboygan
Skunk Haven
Skunk Haven
Timbuktu
Winter Harbor
```

Этот пример демонстрирует, что вы можете объединить несколько последовательностей и отсортировать объединенные данные общим элементом <xsl:perform-sort>. В этом примере первая последовательность

создается элементом <xsl:apply-templates>; встроенные правила таблиц стилей используются для возвращения названий городов. Две другие последовательности создаются на базе серий строковых значений. Комбинация <xsl:perform-sort> и <xsl:sort> создает общую отсортированную последовательность всех значений.

У <xsl:perform-sort> есть одна важная особенность: этот элемент всегда возвращает последовательность. Если определить переменную sortedCities с типом данных xs:string (вместо xs:string*), процессор XSLT выдаст ошибку при отсутствии городов, а также в том случае, если их больше одного.

Наконец, для удаления дубликатов из последовательности можно воспользоваться функцией XPath 2.0 distinct-values():

```
<?xml version="1.0"?>
<!-- perform-sort4.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="sortedCities" as="xs:string*">
      <xsl:perform-sort>
        <xsl:sort select="."/>
        <xsl:apply-templates select="addressbook/address/city"/>
        <xsl:sequence select="('London', 'Adelaide', 'Rome')"/>
        <xsl:sequence select="('Jakarta', 'Sao Paulo', 'Timbuktu')"/>
      </xsl:perform-sort>
    </xsl:variable>
    <xsl:text>Our customers live in these cities:&#xA;&#xA;</xsl:text>
    <xsl:value-of select="distinct-values($sortedCities)"
      separator="&#xA;"/>
  </xsl:template>

</xsl:stylesheet>
```

Теперь славный город Skunk Haven включается в итоговый список только в одном экземпляре:

```
Our customers live in these cities:

Adelaide
Boylston
Jakarta
London
Lynn
Rome
Sao Paulo
Sheboygan
Skunk Haven
Timbuktu
Winter Harbor
```

Обратите внимание: функция distinct-values() вызывается для всей последовательности после того, как она будет сгенерирована. Если вызывать distinct-values() для каждого элемента <xsl:sequence>, то дубликаты будут удалены только из каждой отдельной последовательности.

Группировка узлов

При группировке узлов последовательность сортируется в определенном порядке, в результате чего в ней группируются все объекты с одинаковыми значениями ключа сортировки (или нескольких ключей). Мы воспользуемся xsl:sort для выполнения группировки, а затем завершим работу при помощи переменных и функций (таких как key() или generate-id()).

[2.0] В XSLT 2.0 появились новые элементы и функции, значительно упрощающие группировку. Если вы используете XSLT 2.0, переходите к разделу «[2.0] Новый синтаксис группировки в XSLT 2.0» далее в этой главе.

Первая попытка

В первом примере мы возьмем список адресов и сгруппируем его. Мы найдем все уникальные значения элемента <zip> и сгруппируем адреса по каждому такому значению. Сначала список сортируется по значению <zip>, а затем мы перебираем его значения. Если значение <zip> в текущем объекте не совпадает со значением предыдущего элемента, выводится заголовок, а если они совпадают – выводится адрес. Первая попытка:

```
<?xml version="1.0"?>
<!-- namegrouper1.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Addresses grouped by zip code &#xA;</xsl:text>
    <xsl:for-each select="addressbook/address">
      <xsl:sort select="zip"/>
      <xsl:if test="zip!=preceding-sibling::address[1]/zip">
        <xsl:text>&#xA;Zip code </xsl:text>
        <xsl:value-of select="zip"/>
        <xsl:text> (</xsl:text>
        <xsl:value-of select="city"/>
        <xsl:text>, </xsl:text>
        <xsl:value-of select="state"/>
        <xsl:text>): &#xA;</xsl:text>
      </xsl:if>
      <xsl:if test="name/title">
```

```
        <xsl:value-of select="name/title"/>
        <xsl:text> </xsl:text>
      </xsl:if>
      <xsl:value-of select="name/first-name"/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="name/last-name"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:value-of select="street"/>
      <xsl:text>&#xA;&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Решение состоит из двух шагов:

1. Адреса сортируются по почтовому индексу:

```
<xsl:sort select="zip"/>
```

2. Для каждого адреса, если почтовый индекс не совпадает с предыдущим, выводится заголовок; затем выводятся все адреса, совпадающие с ним:

```
<xsl:if test="zip!=preceding-sibling::address[1]/zip">
  <xsl:text>&#xA;Zip code </xsl:text>
...
```

(Напомню, что `preceding-sibling` **возвращает** `NodeSet/Sequence`**, так что** `preceding-sibling::address[1]` **представляет первый предшествующий узел того же уровня.)**

Вроде бы все разумно, не так ли? Давайте взглянем на результаты:

```
Addresses grouped by zip code

Zip code 00218 (Winter Harbor, ME):
Ms. Natalie Attired
707 Breitling Way

Zip code 02718 (Skunk Haven, MA):
Mary Backstayge
283 First Avenue

Zip code 02718 (Skunk Haven, MA):
Harry Backstayge
283 First Avenue

Zip code 02930 (Lynn, MA):
Ms. Amanda Reckonwith
930-A Chestnut Street

Zip code 27318 (Boylston, VA):
Mary McGoon
103 Bryant Street

Mr. Chester Hasbrouck Frisby
1234 Main Street
```

Да, решение выглядит нормально. У него есть только один серьезный недостаток: *оно не работает*.

В результатах видны две очевидные ошибки: один из адресов (`Mr. Chester Hasbrouck Frisby`) неверно сгруппирован под заголовком `Boylston, Virginia`, а для города `Skunk Haven` создано две группы. Дело в том, что оси работают с *порядком документа*, а не с отсортированным порядком, который мы создали внутри элемента `<xsl:for-each>`.

Как бы просто ни выглядело наше решение, придется поискать другой способ.

Метод «грубой силы»

Можно попытаться провести преобразование в два этапа; сначала напишем промежуточную таблицу стилей для сортировки имен и генерирования нового документа XML, а затем уже написанную таблицу используем с новым документом, в котором порядок документа совпадает с отсортированным порядком. Промежуточная таблица стилей выглядит примерно так:

```
<?xml version="1.0"?>
<!-- namegrouper2a.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" indent="no"/>

  <xsl:strip-space elements="*"/>

  <xsl:template match="/">
    <addressbook>
      <xsl:for-each select="addressbook/address">
        <xsl:sort select="name/zip"/>
        <xsl:copy-of select="."/>
      </xsl:for-each>
    </addressbook>
  </xsl:template>
</xsl:stylesheet>
```

Сгенерированный документ `<addressbook>` содержит все элементы `<address>`, отсортированные в правильном порядке. Далее мы применяем исходную таблицу стилей к отсортированному документу и получаем результат, который уже ближе к истинному:

```
Addresses grouped by zip code
Ms. Natalie Attired
707 Breitling Way

Zip code 02718 (Skunk Haven, MA):
Harry Backstayge
283 First Avenue

Mary Backstayge
```

```
283 First Avenue

Zip code 02930 (Lynn, MA):
Ms. Amanda Reckonwith
930-A Chestnut Street

Zip code 27318 (Boylston, VA):
Mary McGoon
103 Bryant Street

Zip code 48392 (Sheboygan, WI):
Mr. Chester Hasbrouck Frisby
1234 Main Street
```

Но здесь возникает новая проблема: у первой группы (`Winter Harbor`) нет заголовка. Проблема решается просто: нужно слегка изменить выражение XPath и проверить, является ли текущий элемент `<address>` первым:

```
<?xml version="1.0"?>
<!-- namegrouper2b.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text" indent="no"/>

  <xsl:template match="/">
    <xsl:text>Addresses grouped by zip code &#xA;</xsl:text>
    <xsl:for-each select="addressbook/address">
      <xsl:sort select="zip"/>
      <xsl:if test="position() = 1 or
                    zip!=preceding-sibling::address[1]/zip">
        <xsl:text>&#xA;Zip code </xsl:text>
        <xsl:value-of select="zip"/>
...
```

Первый элемент `<address>` не имеет предшествующего одноуровневого узла (`preceding-sibling`). Условие возвращало `false`, поэтому и заголовок у первого адреса не выводился.

Эта пара таблиц стилей работает, но элегантным такое решение не назовешь. Что еще хуже, оно медленно работает – в середине нам приходится останавливаться и записывать файл на диск, а затем снова читать данные с диска. Способ группировки элементов в одной таблице стилей существует, но для этого нам придется воспользоваться другим приемом.

Группировка с использованием <xsl:variable>

Ранее я уже упоминал, что элемент `<xsl:variable>` может пригодиться при выполнении группировки; давайте попробуем им воспользоваться. Мы будем сохранять значение элемента `<zip>` при каждой итерации `<xsl:for-each>`, а `preceding-sibling` будет использоваться несколько иным образом. Попытка номер три выглядит так:

```
<?xml version="1.0"?>
<!-- namegrouper3.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text" indent="no"/>

  <xsl:template match="/">
    <xsl:text>Addresses sorted by zip code&#xA;</xsl:text>
    <xsl:for-each select="addressbook/address">
      <xsl:sort select="zip"/>
      <xsl:variable name="lastZip" select="zip"/>
      <xsl:if test="not(preceding-sibling::address[zip=$lastZip])">
        <xsl:text>Zip code </xsl:text>
        <xsl:value-of select="zip"/>
        <xsl:text>: &#xA;</xsl:text>
        <xsl:for-each select="/addressbook/address[zip=$lastZip]">
          <xsl:sort select="name/last-name"/>
          <xsl:sort select="name/first-name"/>
          <xsl:if test="name/title">
            <xsl:value-of select="name/title"/>
            <xsl:text> </xsl:text>
          </xsl:if>
          <xsl:value-of select="name/first-name"/>
          <xsl:text> </xsl:text>
          <xsl:value-of select="name/last-name"/>
          <xsl:text>&#xA;</xsl:text>
          <xsl:value-of select="street"/>
          <xsl:text>&#xA;&#xA;</xsl:text>
        </xsl:for-each>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Эта таблица стилей выдает желаемый результат:

```
Addresses sorted by Zip Code
Zip code 00218:
Ms. Natalie Attired
707 Breitling Way

Zip code 02718:
Harry Backstayge
283 First Avenue

Mary Backstayge
283 First Avenue

Zip code 02930:
Ms. Amanda Reckonwith
930-A Chestnut Street

Zip code 27318:
Mary McGoon
```

```
103 Bryant Street

Zip code 48392:
Mr. Chester Hasbrouck Frisby
1234 Main Street
```

Почему это решение работает, а первая попытка не работала? Дело в том, что при генерировании вывода мы не полагаемся на отсортированный порядок элементов. Впрочем, у данного решения есть и недостаток: получение желаемых результатов проходит в несколько этапов:

1. Все адреса сортируются по почтовому индексу:

```
<xsl:sort select="zip"/>
```

2. Значение текущего элемента `<zip>` сохраняется в переменной `lastZip`:

```
<xsl:variable name="lastZip" select="zip"/>
```

3. Для каждого элемента `<zip>` мы просматриваем все его предшествующие одноуровневые узлы и проверяем, было ли это конкретное значение (хранящееся в `lastZip`) обнаружено впервые. Если это так, ни одного совпадения среди предшествующих одноуровневых узлов не обнаруживается:

```
<xsl:if test="not(preceding-sibling::address[zip=$lastZip])">
```

4. Если значение в элементе `<zip>` встречается впервые, мы «отступаем» и выбираем заново все элементы `<address>` с потомками `<zip>`, совпадающими с этим значением. После формирования группы элементы сортируются по фамилии/имени с последующим выводом всех адресов.

```
<xsl:for-each select="/addressbook/address[zip=$lastZip]">
  <xsl:sort select="name/last-name"/>
  <xsl:sort select="name/first-name"/>
```

Итак, нам удалось получить желаемые результаты, но весьма неэффективным способом. Мы сортируем данные, затем просматриваем все индексы в отсортированном порядке, проверяем, встречалось ли данное значение в порядке документа, а потом заново выбираем все элементы с совпадающим значением `<zip>` и заново сортируем их перед выводом.

При группировке шести элементов, как в нашем документе, быстродействие останется на достаточном уровне – но с увеличением количества группируемых элементов объем работы, выполняемой процессором XSLT, растет по экспоненте. Должно существовать более эффективное решение, верно? Да, оно существует, и мы рассмотрим его в следующем разделе. Продолжайте читать...

Решение с <xsl:key>

В этом разделе я покажу, как использовать элемент <xsl:key> для группировки данных в документах XML. Этот способ часто называют «методом Мюнха» по фамилии апостола Oracle XML (и автора O'Reilly) Стива Мюнха (Steve Muench), первым предложившего его. Метод Мюнха состоит из трех шагов:

1. Определить ключ для свойства, которое должно использоваться для группировки.
2. Выбрать все группируемые узлы. Для получения набора уникальных значений группировки используется трюк с функциями key() и generate-id().
3. Для каждого уникального значения группировки с помощью функции key() получить все узлы с совпадениями. При необходимости эти узлы можно отсортировать.

Хорошо, это теория; давайте построим таблицу стилей, в которой метод Мюнха будет воплощен на практике. С первым шагом (созданием ключа) все просто. Вот как он выглядит:

```
<xsl:key name="zipcodes" match="address" use="zip"/>
```

Элемент <xsl:key> определяет новый индекс с именем zipcodes, индексирующий элементы <address> по значениям содержащихся в них элементов <zip>.

Теперь, когда ключ определен, можно переходить к более сложной части. Функции key() и generate-id() используются в сочетании друг с другом. Сначала посмотрите, как выглядит синтаксис (ниже он рассматривается во всех подробностях):

```
<xsl:for-each select="//address[generate-id(.)=
  generate-id(key('zipcodes', zip)[1])]">
```

Хорошо, давайте проанализируем этот синтаксис. Мы выбираем все элементы <address>, у которых автоматически сгенерированное значение id совпадает с автоматически сгенерированным значением id первого узла, возвращаемого функцией key(), когда мы запрашиваем все элементы <address> для текущего элемента <zip>.

Все предельно ясно, верно? Давайте попробуем взглянуть на происходящее под немного другим углом.

Для каждого элемента <address> мы вызываем функцию key() для получения всех <address> с одинаковыми <zip>. Затем из полученного набора узлов берется первое значение. Наконец, функция generate-id() генерирует id для обоих узлов. Если два сгенерированных значения id идентичны, значит, это один и тот же узел.

Уф... Дайте перевести дыхание.

Если <address> совпадает с первым узлом, возвращаемым функцией key(), мы знаем, что нашли первый элемент <address>, совпадающий со значением группировки. Выбор всех первых значений (вспомните, что предыдущий предикат завершается [1]) дает набор узлов из некоторого количества элементов <address>, каждый из которых содержит одно из интересующих нас уникальных значений группировки.

Так работает метод Мюнха. На этой стадии мы нашли способ построения набора узлов, содержащих все уникальные значения группировки; теперь необходимо эти узлы обработать. Далее выполняется ряд относительно простых действий:

1. Все узлы сортируются по свойству группировки (элемент <zip> в нашем примере). Сначала мы выбираем первое вхождение каждого уникального элемента <zip> в документе, а затем сортируем эти элементы <zip>. Вот как это выглядит в таблице стилей:

```
<xsl:for-each
  select="//address[generate-id(.)=generate-id(key('zipcodes', zip)[1])]">
  <xsl:sort select="zip"/>
```

2. Внешний элемент <xsl:for-each> выбирает все уникальные значения элемента <zip>. Затем функция key() получает все элементы <address>, соответствующие текущему элементу <zip>:

```
<xsl:for-each select="key('zipcodes', zip)">
```

3. Функция key() возвращает набор узлов из всех подходящих элементов <address>. Каждая группа сортируется по элементам <last-name> и <first-name>, для нее выводится заголовок и все адреса.

Чтобы выходные данные лучше выглядели, в итоговой версии таблицы стилей строится таблица HTML. Полный код таблицы стилей выглядит так:

```
<?xml version="1.0"?>
<!-- namegrouper4.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:key name="zipcodes" match="address" use="zip"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Customers grouped by Zip code</title>
      </head>
      <body style="font-family: sans-serif;">
        <table border="1" cellpadding="5">
          <xsl:for-each select="//address[generate-id(.)=
                                generate-id(key('zipcodes', zip)[1])]">
            <xsl:sort select="zip"/>
```

```
            <xsl:for-each select="key('zipcodes', zip)">
              <xsl:sort select="name/last-name"/>
              <xsl:sort select="name/first-name"/>
              <tr>
                <xsl:if test="position() = 1">
                  <td style="background: #66FF66; text-align: center;
                             vertical-align: middle; font-weight: bold;"
                    rowspan="{count(key('zipcodes', zip))}">
                    <xsl:text>Zip code </xsl:text>
                    <br/>
                    <span style="font-size: 150%;">
                      <xsl:value-of select="zip"/>
                    </span>
                    <br/>
                    <xsl:value-of select="city"/>
                    <xsl:text>, </xsl:text>
                    <xsl:value-of select="state"/>
                  </td>
                </xsl:if>
                <td style="text-align: right; vertical-align: middle;">
                  <xsl:value-of select="name/first-name"/>
                  <xsl:text> </xsl:text>
                  <span style="font-weight: bold; font-size: 125%;">
                    <xsl:value-of select="name/last-name"/>
                  </span>
                </td>
                <td>
                  <xsl:value-of select="street"/>
                </td>
              </tr>
            </xsl:for-each>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Обратите внимание на совместную работу двух элементов `<xsl:for-each>` и разных элементов `<xsl:sort>`. Внешний элемент `<xsl:for-each>` выбирает уникальные значения элемента `<zip>` и сортирует их; внутренний элемент `<xsl:for-each>` выбирает все элементы `<address>`, соответствующие текущему элементу `<zip>`, и сортирует их по значениям `<last-name>` и `<first-name>`.

На рис. 7.1 показано, как сгенерированный документ HTML выглядит в броузере.

В левом столбце для каждого почтового индекса создается ячейка. Это означает, что нам понадобится атрибут `rowspan`, вычисляемый на основании размера текущей группы. Логика вычисления выглядит так:

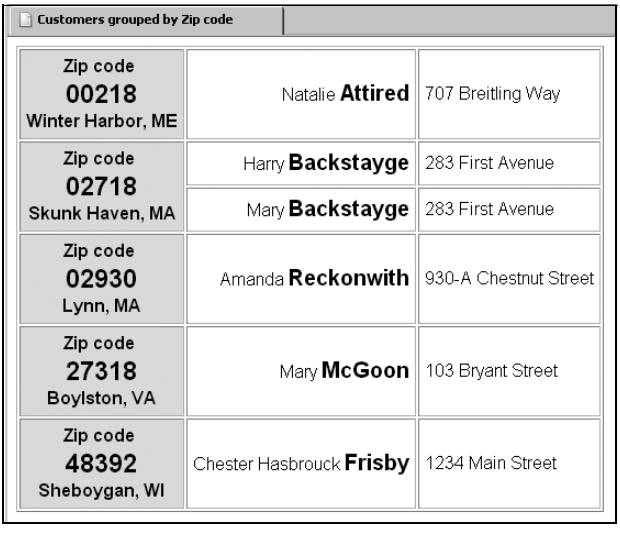


Рис. 7.1. *Документ HTML с группировкой данных*

```
<xsl:if test="position() = 1">
  <td valign="center" bgcolor="#999999"
    rowspan="{count(key('zipcodes', zip))}">
    <b>
      <xsl:text>Zip code </xsl:text>
      <xsl:value-of select="zip"/>
    </b>
  </td>
</xsl:if>
```

[2.0] Новый синтаксис группировки в XSLT 2.0

В 2001 году группа XSL Working Group опубликовала требования к XSLT 2.0 («XSLT Requirements Version 2.0»). Более половины материала находилось под заголовком «Must Simplify Grouping» (Необходимо упростить группировку). (Не представляю, как можно было бы выполнить требование «*Сделать группировку еще более сложной и запутанной*».) Основные изменения в синтаксисе группировки рассматриваются в этом разделе.

Функции группировки XSLT 2.0 строятся на основе элемента `<xsl:for-each-group>`. В этом элементе новые функции XSLT `current-group()` и `current-grouping-key()` используются для работы с группируемыми данными. У элемента `<xsl:for-each-group>` есть четыре взаимоисключающих атрибута, каждый из которых реализует определенный стиль группировки:

`group-by`

Самый распространенный тип группировки. Для идентификации групп используется выражение XPath (например, все элементы `<address>` с одинаковыми кодами `<zip>`), что позволяет организовать перебор содержимого каждой группы.

`group-adjacent`

Этот метод полезен при построении групп, содержащих все смежные узлы, соответствующие выражению XPath. Например, можно взять все смежные элементы `<p>` в документе, преобразовать каждый элемент в элемент `<li>` и заключить всю группу в теги `<ul>` и `</ul>`.

`group-starting-with`

Атрибут `group-starting-with` определяет выражение XPath, идентифицирующее начало группы. Далее каждый последующий элемент добавляется в группу до тех пор, пока не будет обнаружена другая группа. Атрибуты `group-starting-with` и `group-ending-with` чаще всего используются при создании организационной структуры в документах HTML.

`group-ending-with`

Атрибут определяет выражение XPath, идентифицирующее конец группы. При использовании `group-ending-with` элемент `<xsl:for-each-group>` создает новую группу при переходе к обработке узлов. Обнаружив конец группы, процессор XSLT закрывает текущую группу и начинает новую.

Учтите, что все возможности группировки XSLT 2.0 могут быть реализованы и в XSLT 1.0 – просто разметка, которую вам придется написать и сопровождать в XSLT 1.0, значительно усложнится. Если у вас уже имеется рабочая таблица стилей XSLT 1.0, использующая метод Мюнха для выполнения группировки, не стоит изменять ее без надобности.

Самый распространенный стиль группировки: group-by

Как упоминалось ранее, вся группировка в XSLT 2.0 строится на новом элементе `<xsl:for-each-group>`. Этот элемент во многом напоминает `<xsl:for-each>`. В `<xsl:for-each>` при каждой итерации обрабатывается один объект в последовательности; в `<xsl:for-each-group>` при каждой итерации обрабатывается группа. Вернемся к рассмотренному ранее примеру с адресами, в котором каждая группа представляет уникальный почтовый индекс. При использовании `<xsl:for-each-group>` первая группа соответствует значению 00218, вторая – значению 02718 и т. д.

Как говорилось при описании метода Мюнха, для группировки данных в XSLT 1.0 необходимо выполнить три операции:

1. Определить ключ для свойства, которое должно использоваться для группировки.

2. Выбрать все группируемые узлы. Для получения набора уникальных значений группировки используется трюк с функциями `key()` и `generate-id()`.
3. Для каждого уникального значения группировки с помощью функции `key()` получить все узлы с совпадениями. Так как функция `key()` возвращает набор узлов, мы можем провести дальнейшую сортировку набора, соответствующего каждому заданному значению группировки.

В XSLT 2.0 основные операции остаются теми же, но нам не нужно изощряться с `key()` и `generate-id()`. Необходимо решить следующие задачи:

1. Определить выражение XPath для свойства, которое должно использоваться для группировки. Для этого достаточно определить выражение XPath – ключ не нужен.
2. Выбрать все группируемые узлы. Для выбора узлов также используется выражение XPath. Процессор XSLT берет все объекты, соответствующие выражению, и группирует их с использованием определенного нами свойства группировки.
3. Вместо обработки каждого уникального значения свойства, используемого при группировке, мы применяем функцию `current-group()` для обработки каждой группы. Функция `current-group()` возвращает последовательность, поэтому вы можете отсортировать содержимое каждой группы по своему усмотрению. Если понадобится узнать значение ключа группировки, его можно получить функцией `current-grouping-key()`.

Все эти задачи в XSLT 2.0 решаются вполне тривиально. Этапы 1 и 2 реализуются одним элементом `<xsl:for-each-group>`:

```
<xsl:for-each-group select="//address" group-by="zip">
```

В этом примере группируются все элементы `<address>` в документе. Группировка осуществляется по значению элемента `<zip>`.

На третьем этапе функция `current-group()` последовательно обрабатывает каждую группу. Полный код таблицы стилей выглядит так:

```
<?xml version="1.0"?>
<!-- for-each-group_group-by.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" include-content-type="no"/>

  <xsl:template match="/">
    <table border="1" cellpadding="5" style="font-family: sans-serif;">
      <xsl:for-each-group select="//address" group-by="zip">
        <xsl:sort select="current-grouping-key()"/>
        <xsl:for-each select="current-group()">
```

```
        <xsl:sort select="name/last-name"/>
        <xsl:sort select="name/first-name"/>
        <tr>
          <xsl:if test="position() = 1">
            <td valign="center" bgcolor="#999999"
              rowspan="{count(current-group())}">
              <b>
                <xsl:text>Zip code </xsl:text>
                <xsl:value-of select="current-grouping-key()"/>
              </b>
            </td>
          </xsl:if>
          <td align="right">
            <xsl:value-of select="name/first-name"/>
            <xsl:text> </xsl:text>
            <b><xsl:value-of select="name/last-name"/></b>
          </td>
          <td>
            <xsl:value-of select="street"/>
            <xsl:text>, </xsl:text>
            <xsl:value-of select="city"/>
            <xsl:text>, </xsl:text>
            <xsl:value-of select="state"/>
            <xsl:text> </xsl:text>
            <xsl:value-of select="zip"/>
          </td>
        </tr>
      </xsl:for-each>
    </xsl:for-each-group>
  </table>
 </xsl:template>

</xsl:stylesheet>
```

В том, что касается элемента `<xsl:for-each-group>`, некоторые моменты заслуживают особого внимания:

1. Прежде всего элемент `<xsl:sort select="current-grouping-key()"/>` располагает группы в нужном порядке. Элемент `<xsl:sort>` сортирует *группы* на основании их ключей группировки. Порядок следования элементов внутри групп при этом не изменяется.
2. Затем элемент `<xsl:for-each select="current-group">` обеспечивает перебор всех объектов внутри текущей группы. (Напомню, что функция `current-group()` возвращает последовательность узлов, составляющих текущую группу.)
3. Внутри каждой группы данные сортируются по значениям `<last-name>` и `<first-name>`.
4. Остается выполнить последнюю важную операцию: создать первый столбец для отображения текущего ключа группировки. Текущее значение ключа группировки возвращается функцией `current-`

grouping-key(). Для вычисления атрибута rowspan используется шаблон значения атрибута с вызовом функции count(current-group()) для получения количества узлов с текущим ключом группировки.

Сравнение этой таблицы стилей с версией для метода Мюнха показывает, что большая часть кода осталась неизменной; в конце концов, таблица строит тот же документ HTML. Однако существуют некоторые отличия, упрощающие создание и сопровождение таблицы стилей для версии 2.0:

- В таблицах стилей версии 2.0 намного проще указать элемент, используемый для группировки:

```
[2.0] select="//address"

[1.0] select="//address[generate-id(.)=generate-id(key('zipcodes',
                                                    zip)[1])]">
```

- Значительно упрощена работа с данными текущей группы. В таблицах стилей версии 2.0 функции current-group() и current-grouping-key() позволяют работать с текущей группой и значением, которое использовалось для ее создания. В таблице стилей версии 1.0 нам приходилось задавать их каждый раз заново. Обработка всех элементов текущей группы выполняется следующим образом:

```
[2.0] <xsl:for-each select="current-group()">

[1.0] <xsl:for-each select="key('zipcodes', zip)">
```

Или другой пример: определение размера текущей группы тоже значительно упрощается. В таблицах стилей v1.0 приходится переопределять текущую группу при каждой ссылке на нее. Это происходит и при получении количества объектов в текущей группе:

```
[2.0] rowspan="{count(current-group()}"

[1.0] rowspan="{count(key('zipcodes', zip))}"
```

На концептуальном уровне получить значение ключа группировки в таблице стилей v2.0 проще, хотя реальные преимущества от него в этом примере не столь очевидны. (Сравните две таблицы стилей: в v2.0 нам приходится вводить строку <xsl:value-of select="current-grouping-key()"/>, а в v1.0 – только <xsl:value-of select="zip"/>.) Впрочем, при более сложном ключе выигрыш от использования current-grouping-key() был бы более заметным.

Следует также учесть, что при изменении ключа группировки в таблице стилей v1.0 приходится искать все экземпляры ключа группировки и изменять их. В таблице v2.0 ключ группировки определяется в одном месте, а функция current-grouping-key() работает независимо от изменений ключа. Это упрощает сопровождение.

- В v2.0 не нужно определять <xsl:key> отдельно от кода группировки. Это второстепенное обстоятельство, но оно упрощает сопровождение.

Второй тип группировки: group-adjacent

При использовании группировки в режиме `group-adjacent` группа создается на основании элементов, находящихся в смежных позициях исходного документа. В следующем примере используется входной документ HTML, содержащий группы абзацев:

```
<?xml version="1.0"?>
<!-- group-adjacent_input.html -->
<html>
  <body>
    <h2>Steps for grouping in the Muench method</h2>
    <p>Define a <code>key</code> for the property we want
    to use for grouping.</p>
    <p>Select all of the nodes ...</p>
    <p>For each unique grouping value, ...</p>
    <h2>Steps for grouping in XSLT 2.0</h2>
    <p>Define an XPath expression ...</p>
    <p>Select all of the nodes we want to group ...</p>
    <p>Instead of dealing with each ...</p>
  </body>
</html>
```

Перед вами фрагмент текста настоящей главы в формате HTML. (Книга полностью написана в DocBook, номенклатуре XML с четко определенной структурой.) Требуется преобразовать любую последовательность абзацев в неупорядоченный список (`<ul>`), чтобы каждый абзац был преобразован в элемент списка (`<li>`). Для этого мы воспользуемся атрибутом `group-adjacent`. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- for-each-group_group-adjacent.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" include-content-type="no"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Grouping with group-adjacent</title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>Grouping with group-adjacent</h1>
        <xsl:for-each-group select="html/body/*"
          group-adjacent="boolean(self::p)">
          <xsl:choose>
            <xsl:when test="current-grouping-key()">
              <ul>
                <xsl:for-each select="current-group()">
                  <li>
                    <xsl:copy-of select="@*"/>
```

```
                    <xsl:apply-templates select="*|text()"/>
                  </li>
                </xsl:for-each>
              </ul>
            </xsl:when>
            <xsl:otherwise>
              <xsl:for-each select="current-group()">
                <xsl:apply-templates select="."/>
              </xsl:for-each>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:for-each-group>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*"/>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

При использовании `group-adjacent` значение атрибута `group-adjacent` генерирует *атомарное значение* для каждого пункта списка. Иначе говоря, ключ группировки для каждого пункта списка определяется атрибутом `group-adjacent`. В предыдущем примере каждый элемент `<p>` имеет ключ группировки `true`; все остальное имеет ключ группировки `false`. После определения ключа группировки для каждого элемента процессор XSLT 2.0 создает группы. Каждая группа содержит максимальное количество смежных объектов с конкретным значением ключа. Чтобы видеть ключи группировки, в таблицу стилей был включен элемент `<xsl:message>` для вывода сообщения:

```
<xsl:for-each-group select="html/body/*"
  group-adjacent="boolean(self::p)">
  <xsl:message terminate="no">
    <xsl:for-each select="current-group()">
      <xsl:text>current-grouping-key() = </xsl:text>
      <xsl:value-of select="current-grouping-key()"/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:message>
  ...
```

При выполнении таблицы стилей с исходным документом на консоль выводится следующий текст:

```
current-grouping-key() = false

current-grouping-key() = true
```

```
current-grouping-key() = true
current-grouping-key() = true

current-grouping-key() = false

current-grouping-key() = true
current-grouping-key() = true
current-grouping-key() = true
```

Для каждой группы выводится ключ группировки для каждого объекта в этой группе. Как и следовало ожидать, создаются четыре группы. Первая группа содержит элемент <h2> в начале документа, а вторая группа содержит три смежных элемента <p>, следующих за ним. Второй элемент <h2> образует третью группу; четвертая группа состоит из трех смежных элементов <p> в конце. Документ HTML, сгенерированный в результате выполнения таблицы стилей, выглядит так:

```
<html>
  <head>
    <title>Grouping with group-adjacent</title>
  </head>
  <body>
    <h1>Grouping with group-adjacent</h1>
    <h2>Steps for grouping in the Muench method</h2>
    <ul>
      <li>Define a <code>key</code> for the property we want
        to use for grouping.</li>
      <li>Select all of the nodes. ...</li>
      <li>For each unique grouping value, ...</li>
    </ul>
    <h2>Steps for grouping in XSLT 2.0</h2>
    <ul>
      <li>Define an XPath expression ...</li>
      <li>Select all of the nodes we want to group ...</li>
      <li>Instead of dealing with each ...</li>
    </ul>
  </body>
</html>
```

Теперь данные сгруппированы так, как требовалось. Все смежные элементы <p> заменяются неупорядоченным списком, каждый пункт которого представляет элемент <p>. Каждая группа содержит максимальное количество смежных элементов <p>; это позволяет поместить каждую группу внутри элемента <ul>.

Прежде чем переходить к двум последним режимам группировки, мы рассмотрим более сложный пример. Давайте слегка изменим исходный документ HTML и включим в него элементы <p>, которые должны обрабатываться отдельно. Обновленная версия документа HTML выглядит так:

```
<?xml version="1.0"?>
<!-- group-adjacent_input2.html -->
```

```
<html>
  <body>
    <!-- Here's some sample text from the chapter "Sorting
         and Grouping". -->
    <h2>Steps for grouping in the Muench method</h2>
    <p class="item">Define a <code>key</code> for the property we want
    to use for grouping.</p>
    <p class="item">Select all of the nodes ...</p>
    <p class="note">This can be really complicated.</p>
    <p class="note">Many people don't enjoy this method.</p>
    <p class="note">XSLT 2.0 attempts to make grouping simpler.</p>
    <p class="item">For each unique grouping value ...</p>
    <h2>Steps for grouping in XSLT 2.0</h2>
    <p class="item">Define an XPath expression ...</p>
    <p class="item">Select all of the nodes ...</p>
    <p class="note">This is much easier than it used to be.</p>
    <p class="item">Instead of dealing with each ...</p>
  </body>
</html>
```

В новой версии документа элементы `<p>` **делятся на две разновидности. Элементы с атрибутом** `class="item"` **обрабатываются так же, как прежде. Элементы** `class="note"` **будут помещаться в упорядоченный список. Чтобы пример стал еще интереснее, все элементы** `class="note"` **будут последовательно нумероваться в документе. Другими словами, если первая группа смежных элементов** `class="note"` **состоит из трех элементов, нумерация следующей группы элементов** `class="note"` **должна начинаться с 4. Таблица стилей выглядит так (подробное описание приводится ниже):**

```
<?xml version="1.0"?>
<!-- for-each-group_group-adjacent2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" include-content-type="no"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Grouping with group-adjacent</title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>Grouping with group-adjacent</h1>
        <xsl:for-each-group select="html/body/*"
          group-adjacent="if (self::p[@class='item']) then 1
                          else if (self::p[@class='note']) then 2
                          else 3">
          <xsl:choose>

            <!-- Группа для <p class="item"> -->
            <xsl:when test="current-grouping-key() = 1">
```

```
            <ul>
              <xsl:for-each select="current-group()">
                <li>
                  <xsl:copy-of select="@*[not(name()='class')]"/>
                  <xsl:apply-templates select="*|text()"/>
                </li>
              </xsl:for-each>
            </ul>
          </xsl:when>

          <!-- Группа для <p class="note"> -->
          <xsl:when test="current-grouping-key() = 2">
            <xsl:variable name="starting-point">
              <xsl:number count="p[@class='note']"
                level="any" format="1"/>
            </xsl:variable>
            <table border="0" cellpadding="5" width="40%">
              <tr>
                <td width="10%">
                  <p><xsl:text>&#x20;</xsl:text></p>
                </td>
                <td style="background: #CCCCCC;">
                  <p style="font-weight: bold;">
                    <xsl:value-of
                      select="if (count(current-group()) gt 1)
                              then 'Notes'
                              else 'Note'"/>
                  </p>
                  <ol start="{$starting-point}">
                    <xsl:for-each select="current-group()">
                      <li>
                        <xsl:copy-of select="@*[not(name()='class')]"/>
                        <xsl:apply-templates select="*|text()"/>
                      </li>
                    </xsl:for-each>
                  </ol>
                </td>
              </tr>
            </table>
          </xsl:when>

          <!-- Группа для всего остального -->
          <xsl:otherwise>
            <xsl:for-each select="current-group()">
              <xsl:apply-templates select="."/>
            </xsl:for-each>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:for-each-group>
    </body>
  </html>
</xsl:template>
```

```
  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*"/>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Прежде всего обратите внимание на то, что атрибут `group-adjacent` значительно усложнился. Вместо `true` и `false` используются целочисленные значения, потому что количество групп в документе стало больше двух. (Напомню, что ключом группировки `group-adjacent` может быть любое атомарное значение; он не обязано быть равным `true` или `false`.) Элемент `<xsl:for-each-group>` теперь выглядит так:

```
group-adjacent="if (self::p[@class='item']) then 1
                else if (self::p[@class='note']) then 2
                else 3"
```

Таким образом, всем элементам `<p class="item">` назначается ключ группировки 1, всем элементам `<p class="note">` назначается ключ группировки 2, а всем остальным – ключ группировки 3. Включив в таблицу стилей элемент `<xsl:message>`, как это делалось ранее, можно вывести список со значениями ключа группировки:

```
current-grouping-key() = 3

current-grouping-key() = 1
current-grouping-key() = 1

current-grouping-key() = 2
current-grouping-key() = 2
current-grouping-key() = 2

current-grouping-key() = 1
current-grouping-key() = 1

current-grouping-key() = 3

current-grouping-key() = 1
current-grouping-key() = 1

current-grouping-key() = 2

current-grouping-key() = 1
```

Помимо того факта, что теперь используются три группы вместо двух, главные затруднения возникают с обработкой элементов второй группы. В первой группе элементов `<p class="note">` для нумерации можно воспользоваться элементом упорядоченного списка (`<ol>`). Во всех остальных группах придется вычислять исходную точку нумерации. А это означает, что для определения начального номера необходимо найти позицию первого элемента `<p class="note">` по всем элементам `<p class="note">`. Скажем, если группа начинается с 38-го элемента

<p class="note">, в выходном документе HTML необходимо сгенерировать тег <ol start="38">. Элемент <xsl:number> идеально подходит для решения этой задачи.

Для каждой группы элементов <p class="note"> необходимо решить три задачи:

1. Сгенерировать заголовок в зависимости от количества примечаний. Если текущая группа содержит более одного элемента, заголовок имеет вид <h3>Notes</h3>; в противном случае используется заголовок <h3>Note</h3>.
2. Сгенерировать атрибут start для элемента <ol>. Элемент <xsl:number> подсчитывает все элементы <p class="note"> и сообщает позицию первого элемента в группе.
3. Обработать сам элемент, то есть поместить текущий абзац в элемент пункта списка (<li>) и обработать его атрибуты и дочерние элементы.

Итак, начнем с выбора текста заголовка. К счастью, это несложно; мы используем оператор XPath 2.0 if:

```
<p style="font-weight: bold;">
  <xsl:value-of
       select="if (count(current-group()) gt 1)
               then 'Notes'
               else 'Note'"/>
</p>
```

Новый оператор XPath gt сравнивает размер текущей группы – count(current-group()) – с числом 1. Если текущая группа содержит более одного элемента, выбирается текст заголовка Notes, а если нет – текст Note.

Разобравшись с первым пунктом, мы создаем переменную для атрибута start элемента <ol>:

```
<xsl:variable name="starting-point">
  <xsl:number count="p[@class='note']" level="any" format="1"/>
</xsl:variable>
```

Обратите внимание на использование format="1" для создания числового значения. С точки зрения XSLT о типе данных значения можно не беспокоиться (т. е. в элемент <xsl:variable> не нужно включать атрибут as="xs:integer"). Система визуализации HTML интерпретирует это значение как число и использует его соответственно.

На последнем шаге новый элемент <ol> просто направляется в выходной документ (с использованием ранее инициализированной переменной $starting-point), а элемент <p> обрабатывается так же, как прежде:

```
<ol start="{$starting-point}">
  <xsl:for-each select="current-group()">
    <li>
      <xsl:copy-of select="@*[not(name()='class')]"/>
```

```
      <xsl:apply-templates select="*|text()"/>
    </li>
  </xsl:for-each>
</ol>
```

Копируются все атрибуты элемента HTML `<p>`, кроме атрибута `class`, затем с помощью элемента `<xsl:apply-templates>` обрабатывается все, что может находиться в исходном абзаце. Как и в более ранней таблице стилей, шаблон значения атрибута используется для включения атрибута `start` элемента `<ol>`.

На рис. 7.2 показан сгенерированный документ HTML.

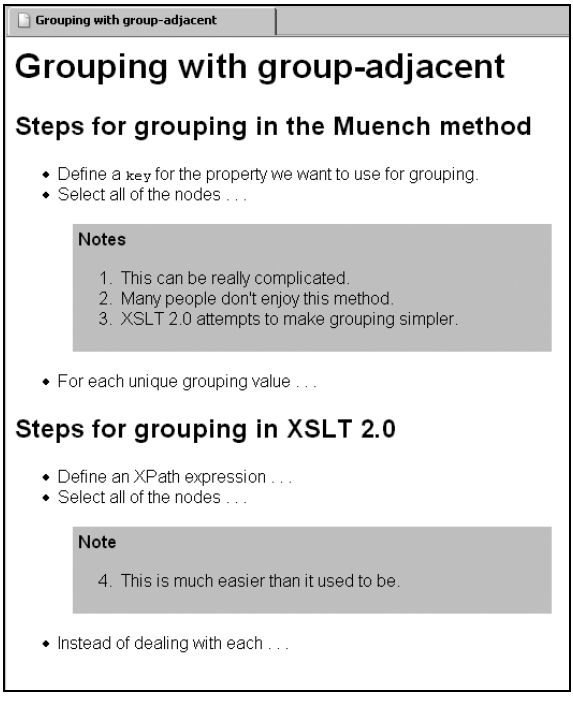


***Рис. 7.2.** Документ HTML с группировкой и сквозной нумерацией данных*

Группировка с помощью group-starting-with

Следующий тип группировки основан на использовании атрибута `group-starting-with`. С этим атрибутом задается узел или условие, начинающее новую группу. Начиная с этого момента, каждый элемент включается в новую группу, пока не будет обнаружен следующий начальный узел или условие. Тогда процессор XSLT закрывает текущую группу и начинает следующую.

Мы воспользуемся group-starting-with для создания организационной структуры в очередном документе HTML. В этом примере новая группа создается для каждого элемента <h1>. Мы используем файл *group-adjacent_input.html* из предыдущего примера.

Разметка преобразуется в формат DocBook, использующий стратегию включения, в отличие от последовательной методологии HTML. Иначе говоря, в HTML заголовок 1 уровня (<h1>) существует сам по себе. Мы можем предполагать, что первый заголовок <h1> начинает раздел, а следующий заголовок <h1> этот раздел завершает, но ничто не гарантирует, что документ HTML структурирован именно так.

В DocBook заголовок уровня 1 является частью раздела, а не автономным элементом. Раздел начинается с тега <sect1> и содержит все вплоть до завершающего тега </sect1>, в том числе и элемент <title> с текстом, который обычно размещается в элементе HTML <h1>.

Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- for-each-group_group-starting-with.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="/">
    <chapter>
      <title>Grouping in XSLT</title>
      <xsl:apply-templates select="html/body"/>
    </chapter>
  </xsl:template>

  <xsl:template match="body">
    <xsl:for-each-group select="*" group-starting-with="h1">
      <sect1>
        <xsl:apply-templates select="current-group()"/>
      </sect1>
    </xsl:for-each-group>
  </xsl:template>

  <xsl:template match="h1">
    <title>
      <xsl:apply-templates/>
    </title>
  </xsl:template>

  <xsl:template match="p">
    <para>
      <xsl:apply-templates/>
    </para>
  </xsl:template>

  <xsl:template match="*">
```

```
    <xsl:copy>
      <xsl:copy-of select="@*"/>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Структурированный результат:

```
<?xml version="1.0" encoding="UTF-8"?>
<chapter>
   <title>Grouping in XSLT</title>
   <sect1>
      <h2>Steps for grouping in the Muench method</h2>
      <para>Define a <code>key</code> for the property we want
    to use for grouping.</para>
      <para>Select all of the nodes ...</para>
      <para>For each unique grouping value, ...</para>
      <h2>Steps for grouping in XSLT 2.0</h2>
      <para>Define an XPath expression ...</para>
      <para>Select all of the nodes we want to group ...</para>
      <para>Instead of dealing with each ...</para>
   </sect1>
</chapter>
```

В сгенерированном коде DocBook каждый элемент `<h1>` становится началом элемента `<sect1>`. Текст `<h1>` становится элементом `<title>` раздела, каждый элемент HTML `<p>` становится элементом DocBook `<para>`, а все остальное копируется в выходной документ в неизменном виде.

Группировка с помощью group-ending-with

Последний способ группировки использует атрибут `group-ending-with`. Он похож на `group-starting-with`, но при группировке в режиме `group-ending-with` определяется условие *завершения* каждой группы. Мы используем `group-ending-with` для размещения данных в таблице HTML, состоящей из трех столбцов. Каждая строка таблицы строится на базе группы; каждая группа содержит не более трех элементов. Для обработки особого случая – строки, содержащей менее трех элементов (`count(current-group()) lt 3`), – будет использоваться упрощенная версия списка машин:

```
<?xml version="1.0"?>
<!-- carlist.xml -->
<cars>
  <make>Alfa Romeo</make>
  <make>Bentley</make>
  <make>Chevrolet</make>
  <make>Dodge</make>
  <make>Eagle</make>
```

```
  <make>Ford</make>
  <make>GMC</make>
  <make>Honda</make>
  <make>Isuzu</make>
  <model>Javelin</model>
  <model>K-Car</model>
  <make>Lincoln</make>
  <make>Mercedes</make>
  <make>Nash</make>
  <make>Opel</make>
  <make>Pontiac</make>
  <model>Quantum</model>
  <model>Rambler</model>
  <make>Studebaker</make>
</cars>
```

Список состоит из 19 машин, так что последняя группа состоит только из одного элемента. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- for-each-group_group-ending-with.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" include-content-type="no"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Car Makes and Models</title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>Car Makes and Models</h1>
        <p>Here are the car makes and models in
        our input document.</p>
        <table border="1" cellpadding="5">
          <xsl:apply-templates select="cars"/>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="cars">
    <xsl:for-each-group select="make|model"
      group-ending-with="*[position() mod 3 = 0]">
      <tr>
        <xsl:apply-templates select="current-group()"/>
        <xsl:if test="count(current-group()) lt 3">
          <td style="background: #CCCCCC;"
            colspan="{3 - count(current-group())}">
          </td>
        </xsl:if>
      </tr>
```

```
    </xsl:for-each-group>
  </xsl:template>

  <xsl:template match="make">
    <td style="font-weight: bold;">
      <xsl:apply-templates/>
    </td>
  </xsl:template>

  <xsl:template match="model">
    <td style="font-style: italic; font-weight: bold;">
      <xsl:apply-templates/>
    </td>
  </xsl:template>

</xsl:stylesheet>
```

Результат показан на рис. 7.3.

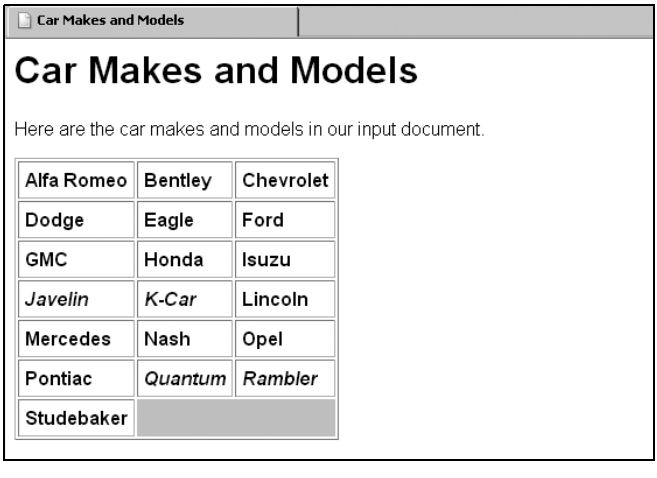


Рис. 7.3. *Документ HTML с группировкой данных по столбцам*

Ключ группировки зависит от позиции каждого элемента. Во входном документе все группируемые данные (элементы `<make>` и `<model>`) находятся на одном уровне документа, поэтому для получения необходимой информации можно воспользоваться функцией `position()`. Функция `position()` возвращает позицию объекта в последовательности. Иначе говоря, элемент `<xsl:for-each-group select="make|model" group-by="*[position() mod 3 = 0]">` создает последовательность, содержащую все элементы `<make>` и `<model>`. Вызов `position()` возвращает позицию объекта в этой последовательности.

На этом наше знакомство с новыми возможностями группировки в XSLT 2.0 подходит к концу. Как я уже говорил, все они могут быть реализованы и в XSLT 1.0, однако синтаксис XSLT 2.0 значительно

упрощает создание и сопровождение таблиц стилей с применением сложной группировки.

Итоги

В этой главе были рассмотрены все стандартные методы сортировки и группировки элементов, включая новые возможности XSLT 2.0. Независимо от того, какие таблицы стилей будут создаваться вами в проектах XML, скорее всего, вам придется использовать средства сортировки и группировки во всем, что вы делаете. В следующей главе речь пойдет о работе с несколькими документами, но эта тема базируется на материале данной главы.

8

Объединение документов

Одной из самых замечательных возможностей XSLT является функция `document()`, позволяющая объединять документы. Можно использовать части документа (заданные при помощи выражений XPath, конечно) для идентификации других документов. Затем вы открываете эти документы и используете функции таблиц стилей с результатом объединения документов. В этой главе мы рассмотрим функцию `document()` во всем блеске ее возможностей.

[2.0] В XSLT 2.0 и XPath 2.0 появились три новые функции для объединения документов: `doc()`, `collection()` и `unparsed-text()`. Функция `doc()`, определяемая XPath 2.0, сходна с функцией `document()`, но обладает меньшими возможностями. Функция `collection()`, также определяемая в XPath 2.0, предназначена для создания коллекций узлов, предоставляемых процессором XSLT. Типы узлов зависят от конкретного процессора. Наконец, функция `unparsed-text()` позволяет включать «неразобранный» текст в данные, обрабатываемые таблицей стилей. В сочетании с такими механизмами, как разбиение строк (tokenization) и регулярные выражения, `unparsed-text()` открывает много новых способов обработки данных. Все эти новые функции описаны в конце главы.

Функция document()

Объединение данных из разных документов входит в число стандартных задач, встречающихся при написании таблиц стилей. Начнем с применения функции `document()` для разбора и обработки нескольких документов XML. В нашем обсуждении будет использоваться документ XML с заказами на покупку следующего вида:

```
<?xml version="1.0"?>
<!-- po38295.xml -->
<purchase-order id="38295">
  <date year="2001" month="9" day="8"/>
  <customer id="4738" level="Basic">
    <address type="business">
      <name>
        <title>Ms.</title>
        <first-name>Amanda</first-name>
        <last-name>Reckonwith</last-name>
      </name>
      <street>930-A Chestnut Street</street>
      <city>Lynn</city>
      <state>MA</state>
      <zip>02930</zip>
    </address>
    <address type="ship-to"/>
  </customer>
  <items>
    <item part-no="23813-03-CDK">
      <name>Cucumber Decorating Kit</name>
      <qty>1</qty>
      <price>29.95</price>
    </item>
  </items>
</purchase-order>
```

При работе с несколькими десятками таких документов часто требуется просмотреть коллекцию заказов в разных представлениях: отсортированными (и даже сгруппированными) по клиенту, по номеру детали, по объему суммарного заказа и т. д. Одно из возможных решений – написание кода, работающего непосредственно с моделью DOM (Document Object Model). Можно разобрать каждый документ, извлечь его дерево DOM, а затем использовать функции DOM для упорядочения и группировки различных деревьев DOM, отображения некоторых частей деревьев DOM и т. д. Но поскольку эта книга посвящена XSLT, вас, вероятно, не удивит, что в XSLT имеется функция, которая выполняет за нас большую часть «черной работы».

Начнем с простого примера использования функции `document()`. Допустим, нам потребовалось объединить несколько заказов в единый документ-отчет. Для этого можно создать главный документ со ссылками на все заказы, включаемые в отчет. Главный документ выглядит примерно так:

```
<?xml version="1.0"?>
<!-- polist.xml -->
<report>
  <title>Selected Purchase Orders</title>
  <po filename="po38292.xml"/>
  <po filename="po38293.xml"/>
```

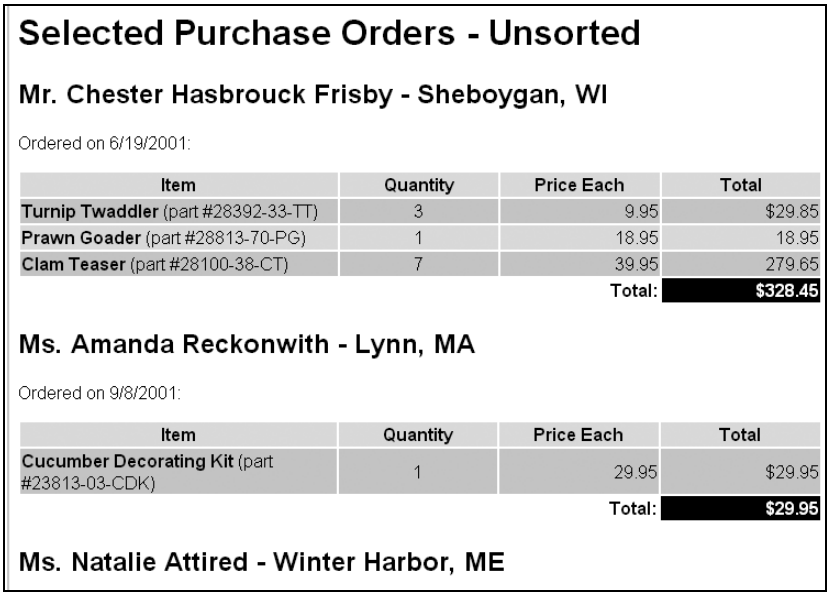
```
  <po filename="po38294.xml"/>
  <po filename="po38295.xml"/>
</report>
```

По мере изложения материала таблица стилей будет постепенно дополняться новыми возможностями, но несколько первых таблиц, которые мы создадим, будут использовать функцию `document()` для открытия нескольких входных файлов. После открытия файлов мы сможем использовать шаблоны в своей таблице стилей для преобразования их содержимого.

В первой таблице стилей в аргументе функции `document()` передается атрибут `filename`. Самое простое, что можно сделать, – открыть каждый заказ и вывести подробную информацию о нем в выходной поток. Центральное место в таблице стилей занимает следующий вызов функции `document()`:

```
<xsl:for-each select="/report/po">
  <xsl:apply-templates
    select="document(@filename)/purchase-order"/>
</xsl:for-each>
```

Результат обработки главного документа с этой таблицей стилей показан на рис. 8.1.



Selected Purchase Orders - Unsorted

Mr. Chester Hasbrouck Frisby - Sheboygan, WI

Ordered on 6/19/2001:

Item	Quantity	Price Each	Total
Turnip Twaddler (part #28392-33-TT)	3	9.95	$29.85
Prawn Goader (part #28813-70-PG)	1	18.95	18.95
Clam Teaser (part #28100-38-CT)	7	39.95	279.65
		Total:	**$328.45**

Ms. Amanda Reckonwith - Lynn, MA

Ordered on 9/8/2001:

Item	Quantity	Price Each	Total
Cucumber Decorating Kit (part #23813-03-CDK)	1	29.95	$29.95
		Total:	**$29.95**

Ms. Natalie Attired - Winter Harbor, ME

Рис. 8.1. *Документ, сгенерированный из нескольких входных файлов*

Главное, на что стоит обратить внимание в этом результате: мы сгенерировали документ, включающий в себя содержимое нескольких других документов. Чтобы пример оставался достаточно коротким, мы объединили всего четыре заказа, однако количество объединяемых документов не ограничено (конечно, если не считать физических ограничений компьютера). Но самое хорошее, что для построения отчета нам не пришлось модифицировать отдельные заказы.

Полная таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- masterdox1.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of select="/report/title"/></title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>Selected Purchase Orders - Unsorted</h1>
        <xsl:for-each select="/report/po">
          <xsl:apply-templates
            select="document(@filename)/purchase-order"/>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="purchase-order">
    <h2>
      <xsl:value-of
        select="customer/address[@type='business']/name/title"/>
      <xsl:text> </xsl:text>
      <xsl:value-of
        select="customer/address[@type='business']/name/first-name"/>
      <xsl:text> </xsl:text>
      <xsl:value-of
        select="customer/address[@type='business']/name/last-name"/>
      <xsl:text> - </xsl:text>
      <xsl:value-of
        select="customer/address[@type='business']/city"/>
      <xsl:text>, </xsl:text>
      <span style="font-weight: bold;">
        <xsl:value-of
          select="customer/address[@type='business']/state"/>
      </span>
    </h2>
    <p>
      <xsl:text>Ordered on </xsl:text>
```

```
    <xsl:value-of select="date/@month"/>
    <xsl:text>/</xsl:text>
    <xsl:value-of select="date/@day"/>
    <xsl:text>/</xsl:text>
    <xsl:value-of select="date/@year"/>
    <xsl:text>:</xsl:text>
  </p>
  <table width="80%" border="0">
    <tr style="background: #66FF66;">
      <th>Item</th>
      <th>Quantity</th>
      <th>Price Each</th>
      <th>Total</th>
    </tr>
    <xsl:for-each select="items/item">
      <tr>
        <xsl:attribute name="style">
          <xsl:text>background: </xsl:text>
          <xsl:choose>
            <xsl:when test="position() mod 2">
              <xsl:text>#CCCCFF</xsl:text>
            </xsl:when>
            <xsl:otherwise>
              <xsl:text>#66FF66</xsl:text>
            </xsl:otherwise>
          </xsl:choose>
          <xsl:text>;</xsl:text>
        </xsl:attribute>
        <td width="40%">
          <span style="font-weight: bold;">
            <xsl:value-of select="name"/>
          </span>
          <xsl:text> (part #</xsl:text>
          <xsl:value-of select="@part-no"/>
          <xsl:text>)</xsl:text>
        </td>
        <td style="text-align: center;" width="20%">
          <xsl:value-of select="qty"/>
        </td>
        <td style="text-align: right;" width="20%">
          <xsl:value-of select="price"/>
        </td>
        <td style="text-align: right;" width="20%">
          <xsl:choose>
            <xsl:when test="position()=1">
              <xsl:value-of
                select="format-number(price * qty, '$#,###.00')"/>
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of
                select="format-number(price * qty, '#,###.00')"/>
```

```
            </xsl:otherwise>
          </xsl:choose>
        </td>
      </tr>
    </xsl:for-each>
    <tr style="font-weight: bold;">
      <td colspan="3" style="text-align: right;">
        Total:
      </td>
      <td style="text-align: right; color: white; background: black;">
        <xsl:variable name="orderTotal">
          <xsl:call-template name="sumItems">
            <xsl:with-param name="items" select="items/item" />
          </xsl:call-template>
        </xsl:variable>
        <xsl:value-of
          select="format-number($orderTotal, '$#,###.00')"/>
      </td>
    </tr>
  </table>
</xsl:template>

<xsl:template name="sumItems">
  <xsl:param name="items" />
  <xsl:param name="runningTotal" select="0" />
  <xsl:choose>
    <xsl:when test="$items">
      <xsl:variable name="firstItemSubtotal"
        select="$items[1]/qty * $items[1]/price" />
      <xsl:call-template name="sumItems">
        <xsl:with-param name="items" select="$items[position() > 1]" />
        <xsl:with-param name="runningTotal"
          select="$runningTotal + $firstItemSubtotal" />
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$runningTotal" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Шаг в сторону: рекурсия при математических вычислениях

Здесь же мне хотелось бы ненадолго остановиться на рекурсии, применяемой для вычисления общей суммы по каждому заказу. На первый взгляд кажется, что данный случай идеально подходит для использования функции `sum()`. Общая сумма заказа вычисляется суммированием

цены по каждой позиции, умноженной на количество заказанных единиц. Можно попытаться использовать вызов sum() следующего вида:

```
<xsl:value-of select="sum(item/qty*item/price)"/>
```

К сожалению, функция sum() просто получает переданный ей набор узлов, преобразует каждый узел из набора в число, а затем возвращает сумму полученных чисел. Выражение item/qty*item/price является действительным выражением XPath, но не является действительным набором узлов. С учетом этого обстоятельства нам пришлось создать рекурсивный шаблон <xs:template>, который выполнит работу за нас. В его реализации используется пара приемов, заслуживающих внимания; мы разберем эти приемы, когда речь зайдет об их использовании в таблице стилей.

[2.0] Новые функции XPath 2.0 позволяют обойтись без рекурсии. В XSLT 2.0 эта таблица стилей записывается намного проще и короче; подробности изложены в последнем разделе этой главы. Если вы используете процессор XSLT 2.0, а тема рекурсии вас не интересует, переходите к следующему разделу.

Рекурсивная реализация

Все начинается с передачи рекурсивному шаблону набора всех элементов <item>:

```
<xsl:variable name="orderTotal">
  <xsl:call-template name="sumItems">
    <xsl:with-param name="items" select="items/item" />
  </xsl:call-template>
</xsl:variable>
```

Значение, возвращаемое рекурсивным шаблоном с именем sumItems, становится значением переменной $orderTotal. Внутри рекурсивного шаблона используются два параметра: переменная, содержащая все элементы <item>, и сумма всех объектов, обработанных до настоящего момента. Конечно, при первом вызове шаблона значение последнего равно нулю. (Параметру задается значение по умолчанию, чтобы нам не пришлось беспокоиться о нем при первом вызове шаблона.) Два параметра шаблона:

```
<xsl:template name="sumItems">
  <xsl:param name="items" />
  <xsl:param name="runningTotal" select="0" />
```

Тело рекурсивного шаблона представляет собой элемент <xsl:choose>. Если переменная $items содержит хотя бы один узел, мы вычисляем промежуточный итог для текущего товара (цена единицы, умноженная на количество):

```
<xsl:when test="$items">
  <xsl:variable name="firstItemSubtotal"
    select="$items[1]/qty * $items[1]/price" />
```

Вернемся к правилам преобразования переменных в логические значения: набор узлов (или последовательность в XSLT 2.0), начинающийся, как минимум, с одного узла, интерпретируется как `true`. Также обратите внимание на то, что промежуточный итог вычисляется только для первого объекта (`$items[1]`). После вычисления промежуточного итога для текущего товара шаблон вызывает сам себя:

```
<xsl:call-template name="sumItems">
  <xsl:with-param name="items" select="$items[position() > 1]" />
  <xsl:with-param name="runningTotal"
    select="$runningTotal + $firstItemSubtotal" />
</xsl:call-template>
```

Обратите внимание: параметр `$items`, передаваемый при следующем вызове шаблона, содержит все объекты, следующие после первого (`position() > 1`). Параметр `$runningTotal` содержит сумму предыдущего значения параметра и промежуточного итога по текущей позиции.

Шаблон вызывает сам себя до тех пор, пока параметр `$items` не станет пустым. Это означает, что атрибут `test="$items"` вернет `false` и будет выбрана ветвь `<xsl:otherwise>` элемента `<xsl:choose>`. Эта ветвь использует элемент `<xsl:value-of>` для вывода окончательного значения. (Другие вызовы `sumItems` ничего не выводят.)

Полный рекурсивный шаблон выглядит так:

```
<xsl:template name="sumItems">
  <xsl:param name="items" />
  <xsl:param name="runningTotal" select="0" />
  <xsl:choose>
    <xsl:when test="$items">
      <xsl:variable name="firstItemSubtotal"
        select="$items[1]/qty * $items[1]/price" />
      <xsl:call-template name="sumItems">
        <xsl:with-param name="items" select="$items[position() > 1]" />
        <xsl:with-param name="runningTotal"
          select="$runningTotal + $firstItemSubtotal" />
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$runningTotal" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Рекурсия часто применяется в таблицах стилей XSLT 1.0. В XSLT 2.0 во многих ситуациях удается обойтись без рекурсии (подробности чуть позже), однако в отдельных случаях рекурсия остается единственным способом решения некоторых проблем в XSLT. Навыки использования рекурсии вам пригодятся.

Применение функции format-number() для управления выводом

В приведенной таблице стилей также стоит обратить внимание на использование функции XSLT `format-number()` при отображении общей суммы текущего заказа. Ранее мы уже обсудили, как задать переменной `$orderTotal` вывод шаблона `sumItems`. После того как значение переменной будет задано, функция `format-number` используется для его вывода со знаком денежной единицы, запятыми и двумя цифрами в дробной части:

```
<xsl:value-of select="format-number($order-total, '$#,###.00')"/>
```

Базовые URI-адреса и функция document()

В предшествующей таблице стилей функция `document()` использовалась для выборки некоторого количества узлов исходного документа (списка заказов) и открытия соответствующих файлов. Функция `document()` может вызываться несколькими разными способами, кратко описанными ниже.

Самый распространенный вариант вызова `document()` – тот, который мы только что использовали. Выражение XPath описывает набор узлов; функция `document()` берет каждый узел набора, преобразует его в строку и рассматривает эту строку как URI-адрес. Таким образом, когда мы передаем набор узлов с атрибутами `filename` из списка заказов, каждый узел используется как URI-адрес. Если URI-адреса представляют собой относительные ссылки (т. е. не начинаются с имени протокола, например `http`), процессору XSLT необходим базовый URI-адрес. Если аргумент является узлом, базовый URI-адрес узла используется для разрешения относительной ссылки. Если аргумент является строкой, как в нашем случае, используется базовый URI-адрес таблицы стилей.

Каждый узел исходного дерева XPath ассоциируется с *базовым URI-адресом*. При использовании функции `document()` базовый URI-адрес необходим для разрешения ссылок на различные ресурсы, заданные в формате относительных ссылок.

В этой врезке базовые URI-адреса рассматриваются более подробно, чем вам, скорее всего, когда-нибудь понадобится. Если какой-либо узел является узлом элемента или узлом инструкции по обработке и он находится во внешней сущности, то базовый URI-адрес этого узла является базовым URI-адресом внешней сущности. Если элемент или инструкция по обработке не принадлежит внешней сущности, то его базовый URI-адрес является базовым URI-адресом документа, в котором он находится. Базовый URI-адрес узла документа является URI-адресом самого документа, а базовый URI-адрес узла атрибута, комментария, пространства имен или текста является базовым URI-адресом родителя этого узла.

> Для задания базового URI-адреса узла в исходном документе XML используется атрибут `xml:base`. Если заданный узел не имеет базового URI-адреса, процессор XSLT проверяет предков узла (родителя узла, затем родителя родителя и т. д.), пока не найдет атрибут `xml:base`. Если ни узел, ни его предки не имеют атрибута `xml:base`, используется базовый URI-адрес таблицы стилей. За дополнительной информацией о базовых URI-адресах обращайтесь к определению функции `base-uri()`.

Если функция `document()` вызывается с двумя аргументами, второй аргумент должен быть набором узлов. Первый аргумент обрабатывается по только что описанным правилам, а второй аргумент задает базовый URI-адрес – им является базовый URI-адрес первого узла в наборе (или последовательности в XSLT 2.0). Форма `document()` с двумя аргументами встречается относительно редко, но при необходимости ее можно использовать.

В нашем примере используются строки без указания конкретного протокола (`po38293.xml` вместо `http://.../po38293.xml`). Функции `document()` можно также передать URL. Чтобы открыть конкретный ресурс, достаточно указать его имя:

```
document('http://www.ibm.com/developerworks/news/dw_dwtp.rss')
```

Этот вызов функции открывает указанный ресурс и обрабатывает его. Учтите, что в случае неудачи при поиске ресурса процессор XSLT должен либо сообщить об ошибке и прервать работу, либо вернуть пустой набор узлов. Процессоры XSLT также не обязаны поддерживать конкретные протоколы (`http`, `ftp` и т. д.); за информацией о поддерживаемых протоколах обращайтесь к документации своего процессора XSLT.

Наконец, существует особая ситуация с передачей `document()` пустой строки. Встречая вызов `document('')`, процессор XSLT разбирает текущую таблицу стилей и возвращает один узел – корневой узел самой таблицы стилей. Этот прием часто используется при обработке таблиц отображения, о чем будет рассказано позднее в этой главе.

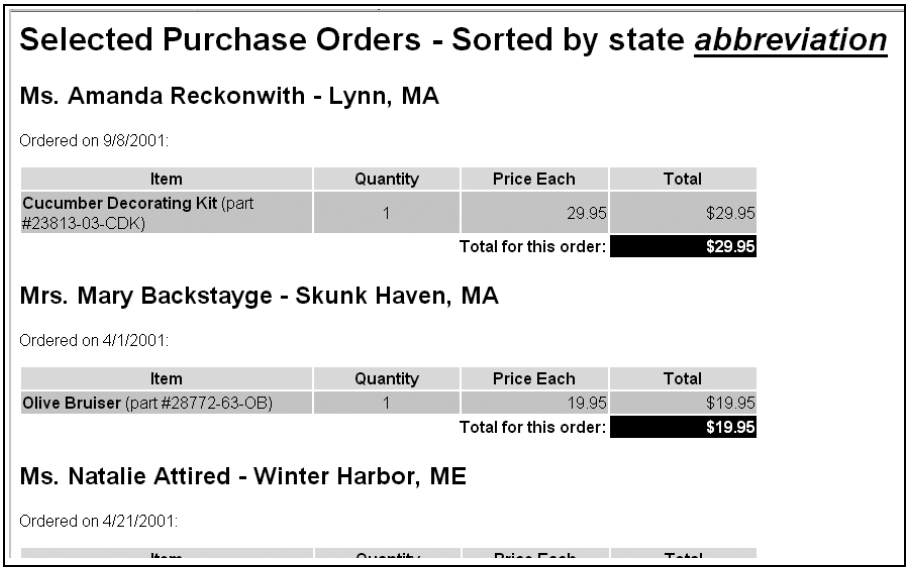
Функция document() и сортировка

Пока что мы написали простой документ XML, содержащий ссылки на другие документы XML, а затем создали таблицу стилей, которая объединяет все документы XML, упоминаемые в ссылках, в один выходной документ. Все это, конечно, хорошо... Но, скорее всего, на практике вам потребуется сделать что-нибудь менее тривиальное: например, сгенерировать документ с перечислением всех товаров, присутствующих во всех заказах, или отсортировать заказы по штату или по фамилии клиента. Мы рассмотрим некоторые из этих сценариев для демонстрации типичных задач, встречающихся при построении документов из нескольких входных файлов.

Итак, первая задача – сгенерировать список всех заказов и отсортировать его сначала по штату, а затем по городам внутри штата. Особо сложной ее никак не назовешь; достаточно использовать элемент `<xsl:sort>` в сочетании с функцией `document()`. Основная часть таблицы стилей выглядит так:

```
<?xml version="1.0"?>
<!-- masterdox2.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
...
<xsl:apply-templates
  select="document(/report/po/@filename)/purchase-order">
  <xsl:sort select="customer/address/state"/>
  <xsl:sort select="customer/address/city"/>
</xsl:apply-templates>
```

Мы выбираем все элементы `<purchase-order>` и сортируем их по значениям элементов `<state>` и `<city>`. На рис. 8.2 показан выходной документ с заказами, отсортированными по элементу `<state>` (содержащему сокращенное обозначение штата).



Selected Purchase Orders - Sorted by state *abbreviation*

Ms. Amanda Reckonwith - Lynn, MA

Ordered on 9/8/2001:

Item	Quantity	Price Each	Total
Cucumber Decorating Kit (part #23813-03-CDK)	1	29.95	$29.95
		Total for this order:	**$29.95**

Mrs. Mary Backstayge - Skunk Haven, MA

Ordered on 4/1/2001:

Item	Quantity	Price Each	Total
Olive Bruiser (part #28772-63-OB)	1	19.95	$19.95
		Total for this order:	**$19.95**

Ms. Natalie Attired - Winter Harbor, ME

Ordered on 4/21/2001:

Рис. 8.2. *Заказы, отсортированные по сокращенному названию штата*

Обратите внимание: сортировка осуществляется по *сокращению*, а не по фактическому названию штата; эта проблема будет решена в следующем примере.

Реализация таблиц отображения

Как упоминалось ранее, вызов функции document() с пустой строкой позволяет обращаться к узлам самой таблицы стилей. Мы можем воспользоваться этим обстоятельством для реализации таблицы отображения. В качестве примера будет создана таблица, связывающая сокращение вида ME с полным названием штата (Maine). Далее значение из таблицы отображения может использоваться в качестве ключа сортировки. Внимательные читатели могли заметить, что в предыдущем примере после сортировки сокращение MA находится перед сокращением ME, однако в отсортированном списке штат Мэн (Maine, сокращение ME) оказывается перед Массачусетсом (Massachusetts, сокращение MA).

Начнем с создания таблицы отображения. Мы воспользуемся тем фактом, что элементом верхнего уровня в таблице стилей может быть произвольный элемент – при условии, что в этом элементе уточнено пространство имен, отличное от пространства имен xsl:, зарезервированного для таблиц стилей. Определение префикса пространства имен и часть таблицы отображения, в которой этот префикс используется, выглядят так:

```
<?xml version="1.0"?>
<!-- masterdox3.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:states="http://www.usps.com/ncsc/lookups/abbreviations.html"
  exclude-result-prefixes="states">

  <states:name abbrev="AL">Alabama</states:name>
  <states:name abbrev="AK">Alaska</states:name>
  <states:name abbrev="AS">American Samoa</states:name>
  <!-- Many state names deleted for brevity -->
  <states:name abbrev="WV">West Virginia</states:name>
  <states:name abbrev="WI">Wisconsin</states:name>
  <states:name abbrev="WY">Wyoming</states:name>
```

(Префикс states отображается на URL-адрес официального списка сокращенных названий штатов Почтовой службы США, хотя вместо него может быть задана произвольная строка.)

Чтобы организовать поиск соответствия по таблице, мы воспользуемся функцией document() для получения корневого узла нашей таблицы стилей, а затем проведем поиск элемента <states:name> с атрибутом abbrev, совпадающим со значением текущего элемента <state> в заказе, обрабатываемом в данный момент. Все это волшебство реализуется следующей (несколько хитроумной) синтаксической конструкцией:

```
<body style="font-family: sans-serif;">
  <h1>
    Selected Purchase Orders - Sorted by state
    <span style="font-style: italic;
```

```
                text-decoration: underline;">name</span>
    </h1>
    <xsl:for-each
      select="document(/report/po/@filename)/purchase-order
              /customer/address/state">
      <xsl:sort
        select="document('')/*/states:name[@abbrev=current()]"/>
      <xsl:sort select="../city"/>
      <xsl:apply-templates select="ancestor::purchase-order"/>
    </xsl:for-each>
  </body>
```

Обратите внимание на то, что функция `document()` используется дважды: для открытия документа, на который ссылается атрибут `filename`, и для открытия самой таблицы стилей. Выражение XPath в атрибуте `select` элемента `<xsl:sort>` также заслуживает более подробного рассмотрения. Оно состоит из четырех важнейших частей:

`document('')`

Возвращает корневой узел текущей таблицы стилей.

`/*/`

Указывает, что последующая конструкция должна быть элементом верхнего уровня таблицы стилей. Синтаксис начинается с корня документа и состоит из одного элемента. Для текущей таблицы стилей выражение XPath можно было записать в следующем виде:

```
select="document('')/xsl:stylesheet/
        states:name[@abbrev=current()]"
```

Так как корневым элементом таблицы стилей может быть как `<xsl:stylesheet>`, так и `<xsl:transform>`, лучше применять звездочку.

`states:name`

Мы ищем элемент `name` в пространстве имен `states:`.

`[@abbrev=current()]`

Означает, что атрибут `abbrev` текущего элемента `<states:name>` обладает таким же значением, как и текущий узел. Нам приходится использовать функцию XSLT `current()`, потому что нам нужен текущий, а не контекстный узел. Внутри предикатного выражения текущим узлом является обрабатываемый элемент `<state>`, тогда как контекстным узлом является элемент `<states:name>`, содержащий проверяемый атрибут `abbrev`.

Обратите внимание: атрибут `<xsl:stylesheet>` содержит атрибут `exclude-result-prefixes`, который гарантирует, что в выходном документе не объявляется префикс `states:`. Элементы этого пространства имен используются только для реализации таблицы отображения; конечно, в выходном документе HTML пространство имен `states:` определяться не должно.

Как и в любой таблице стилей, выражение XPath можно записать несколькими разными способами. Один из вариантов обращения к таблице отображения выглядит так:

```
<xsl:for-each
  select="document(/report/po/@filename)/purchase-order
          /customer/address/state">
  <xsl:sort
    select="document('')/*/states:name[@abbrev=current()]"/>
  <xsl:sort select="../city"/>
  <xsl:apply-templates select="ancestor::purchase-order"/>
</xsl:for-each>
```

Другой возможный вариант:

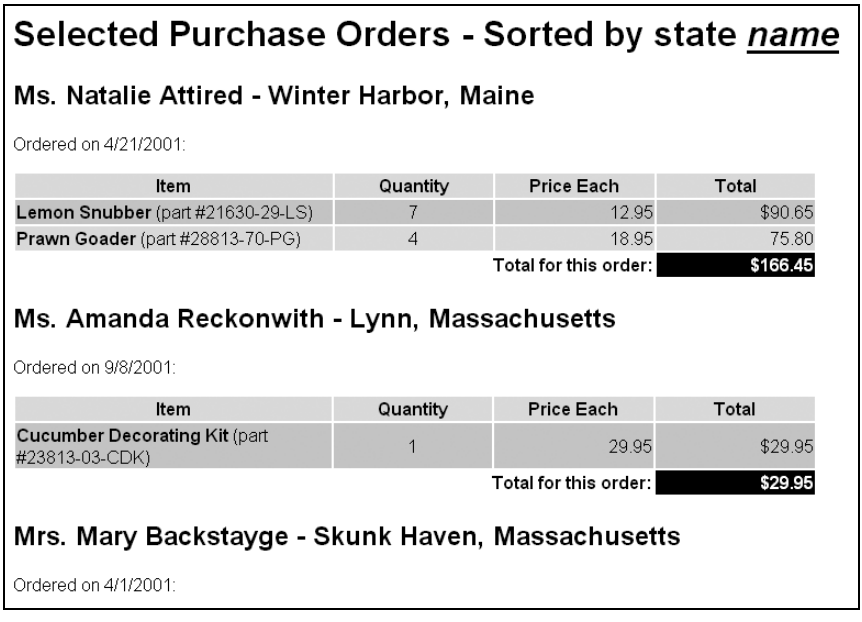
```
<xsl:for-each
  select="document(/report/po/@filename)/purchase-order">
  <xsl:sort
    select="document('')/*/states:name
            [@abbrev=current()/customer/address/state]"/>
  <xsl:sort select="customer/address/city"/>
  <xsl:apply-templates select="."/>
</xsl:for-each>
```

Оба решения делают одно и то же, но по-разному задают части документа. Первый пример создает цикл `<xsl:for-each>` для всех элементов `<state>`; во втором используются элементы `<purchase-order>`. В зависимости от этого изменяется отправная точка в дереве узлов для написания других выражений XPath. Для элемента `<state>` местоположение соответствующего элемента `<city>` определяется выражением `../city`, и для обработки `<purchase-order>` приходится использовать ось `ancestor`. Если же `<xsl:for-each>` будет базироваться на элементах `<purchase-order>`, элемент `<city>` задается записью `customer/address/city`. Во втором примере `<purchase-order>` задается точкой.

Оба примера почти не отличаются по сложности, но вы всегда должны стремиться к упрощению записи выражений XPath. Если все выражения XPath в элементе `<xsl:for-each>` или `<xsl:template>` оказываются очень сложными, возможно, вам удастся изменить внешнее выражение и упростить таблицу стилей.

На рис. 8.3 показан вывод, полученный от таблицы стилей с таблицей отображения.

На этот раз заказы сортируются по фактическому названию штата, а не по сокращению. Таблицы отображения можно рассматривать как чрезвычайно полезный побочный эффект формы `document('')`. Таблицу отображения можно поместить в другом файле и загрузить данные из другого файла функцией `document()`, однако методика, примененная в нашем примере, является самой распространенной формой реализации таблиц отображения.



Selected Purchase Orders - Sorted by state *name*

Ms. Natalie Attired - Winter Harbor, Maine

Ordered on 4/21/2001:

Item	Quantity	Price Each	Total
Lemon Snubber (part #21630-29-LS)	7	12.95	$90.65
Prawn Goader (part #28813-70-PG)	4	18.95	75.80
		Total for this order:	**$166.45**

Ms. Amanda Reckonwith - Lynn, Massachusetts

Ordered on 9/8/2001:

Item	Quantity	Price Each	Total
Cucumber Decorating Kit (part #23813-03-CDK)	1	29.95	$29.95
		Total for this order:	**$29.95**

Mrs. Mary Backstayge - Skunk Haven, Massachusetts

Ordered on 4/1/2001:

Рис. 8.3. *Заказы, отсортированные по названию штата*

Группировка по нескольким документам

Остается решить последнюю задачу – сгруппировать коллекцию заказов. Мы создадим новую таблицу, в которой заказы группируются по штату, в который производится поставка. При этом будет использоваться методика группировки, использовавшаяся ранее в главе 7.

Мы определим ключ XSLT `key()` для всех значений `<state>` в нашем наборе, а также создадим переменную, в которой будут храниться заказы. Хранение всех заказов в переменной позволит избежать вызова функции `document()` каждый раз, когда вам потребуется обратиться к конкретному заказу. Когда эти две структуры будут готовы, можно переходить к реализации группировки по принципам, представленным в предыдущей главе.

Начнем с определения ключа:

```
<xsl:key name="po-key" match="purchase-order"
  use="customer/address/state"/>
```

А вот как происходит загрузка всех документов с заказами в переменную:

```
<xsl:variable name="purchase-orders"
  select="document(/report/po/@filename)/purchase-order"/>
```

Итак, мы получили переменную, содержащую все узлы из всех заказов, и мы можем переходить к обработке данных. В ходе обработки будут выполняться следующие действия:

1. Создать элемент <xsl:for-each> для обработки всех узлов в переменной $purchase-orders.
2. Отсортировать заказы по названию штата. Как и в предыдущей таблице стилей, для этого функция document('') получает название штата, соответствующее сокращению в заказе.
3. Сохранить значение текущего элемента <state> в переменной.
4. Использовать функцию key() для сохранения в переменной всех заказов из текущего штата.
5. Если данный штат встречается впервые (здесь используется функция generate-id()), элемент <xsl:apply-templates> обрабатывает все заказы для текущего штата. При вызове <xsl:apply-templates> заказы, относящиеся к этому штату, сортируются по городу перед обработкой.

Далее приводится содержательная часть таблицы стилей; шаблон для обработки элементов <purchase-order> не изменился по сравнению с предыдущими таблицами:

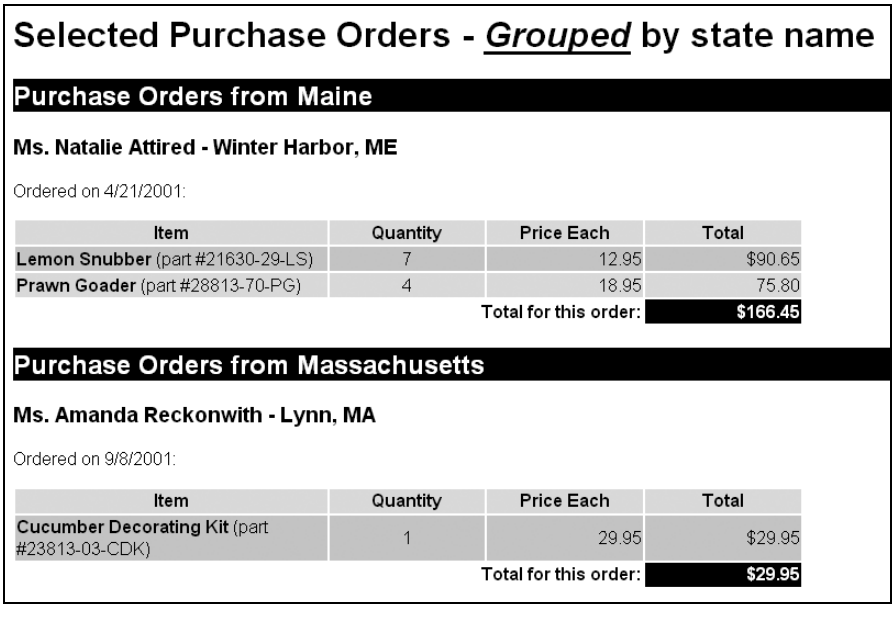
```
<?xml version="1.0"?>
<!-- masterdox4.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:states="http://www.usps.com/ncsc/lookups/abbreviations.html"
  exclude-result-prefixes="states">

...

<xsl:for-each select="$purchase-orders">
  <xsl:sort
    select="document('')/*/states:name
            [@abbrev=current()/customer/address/state]"/>
  <xsl:variable name="currentState"
    select="customer/address/state"/>
  <xsl:variable name="currentStatePOs"
    select="$purchase-orders/key('po-key', $currentState)"/>
  <xsl:if
    test="generate-id() = generate-id($currentStatePOs[1])">
    <h2 style="color: white; background: black;">
      Purchase Orders from
      <xsl:value-of
        select="document('')/*/states:name
                [@abbrev=$currentState]"/>
    </h2>
    <xsl:apply-templates select="$currentStatePOs">
      <xsl:sort select="customer/address/city"/>
    </xsl:apply-templates>
  </xsl:if>
</xsl:for-each>
```

Обратите внимание на использование функции `key()` для выборки всех элементов `<purchase-order>`, относящихся к текущему штату. Все эти заказы сохраняются в переменной. Чтобы определить, встречается ли заказ впервые, мы генерируем идентификатор для текущего заказа (`generate-id()`) и сравниваем его со сгенерированным идентификатором первого объекта в наборе заказов из текущего штата (`generate-id($currentStatePOs[1])`). Если два сгенерированных идентификатора совпадают, значит, название штата встречается впервые.

Результат всех этих усилий показан на рис. 8.4.



Selected Purchase Orders - *Grouped* by state name

Purchase Orders from Maine

Ms. Natalie Attired - Winter Harbor, ME

Ordered on 4/21/2001:

Item	Quantity	Price Each	Total
Lemon Snubber (part #21630-29-LS)	7	12.95	$90.65
Prawn Goader (part #28813-70-PG)	4	18.95	75.80
		Total for this order:	**$166.45**

Purchase Orders from Massachusetts

Ms. Amanda Reckonwith - Lynn, MA

Ordered on 9/8/2001:

Item	Quantity	Price Each	Total
Cucumber Decorating Kit (part #23813-03-CDK)	1	29.95	$29.95
		Total for this order:	**$29.95**

Рис. 8.4. *Заказы, сгруппированные по названию штата*

[2.0] Упрощение таблиц стилей в XSLT 2.0

В этой главе неоднократно упоминалось о том, что XSLT 2.0 обладает возможностями, значительно упрощающими таблицы стилей. Соответствующие возможности рассматриваются в этом разделе.

Созданная нами таблица стилей XSLT 1.0 может быть усовершенствована в нескольких аспектах:

- Нам приходится использовать неудобную методику группировки XSLT 1.0. Однако этого можно избежать, воспользовавшись элементом XSLT 2.0 `<xsl:for-each-group>` для выполнения группировки. Процессор XSLT берет на себя поиск уникальных значений названия штата, а это значительно упрощает нашу работу.

- Для вычисления промежуточного итога по каждому заказу применяется рекурсия. Новые возможности XPath 2.0 вычисляют промежуточный итог в одном простом выражении.
- Хотя описанный ранее способ с `document('')` работает, такое решение получается запутанным и неестественным. Его можно заменить функцией XSLT, которая получает сокращение штата и возвращает его полное имя. Синтаксис при этом существенно упрощается. Вызов `getStateName()` понять и сопровождать намного проще, чем строку `document('')/*/states:name[@abbrev=$next-state]`.
- Второстепенный момент: в паре мест нам приходится применять элемент `<xsl:choose>`. Вместо него можно воспользоваться оператором `if`, определенным в спецификации XPath 2.0 and XQuery 1.0 Functions and Operators; это также приведет к упрощению таблицы стилей. Элементы `<xsl:choose>`, `<xsl:when>` и `<xsl:otherwise>` заменяются на `if`, `then` и `else`.
- Еще один второстепенный момент: дата из каждого заказа хранится в исходном документе XML в виде набора из трех атрибутов. Мы можем использовать поддержку типов данных XML Schema из XSLT 2.0 для создания `xs:date`, а затем при помощи мощной функции `format-date()` более элегантно отформатировать дату.

Приняв во внимание все сказанное, мы переходим к упрощению таблицы стилей.

Группировка по уникальным значениям

Один из недостатков итоговой таблицы стилей XSLT 1.0 заключается в том, что нам приходится искать все уникальные штаты во всех заказах. Задача решается методом Мюнха, однако это усложняет написание и сопровождение кода. Элемент `<xsl:for-each-group>` позволяет существенно упростить программирование. С атрибутом `group-by` элемент `<xsl:for-each-group>` автоматически находит все уникальные значения элемента `<state>`. Соответствующий фрагмент кода выглядит так:

```
<xsl:for-each-group
  select="document(/report/po/@filename)/purchase-order"
  group-by="customer/address/state">
  <xsl:sort
    select="states:getStateName(current-grouping-key())"/>
```

Элемент `<xsl:for-each-group>` группирует все элементы `<purchase-order>` по значениям их элементов `<state>`. Элемент `<xsl:sort>` сортирует группы по названиям штатов: `current-grouping-key()` возвращает значение текущего элемента `<state>`. Запись `states:getStateName()` обозначает функцию XSLT (см. далее в разделе «Реализация таблиц отображения с использованием <xsl:function>»).

Сравним эту таблицу стилей с нашей итоговой таблицей стилей для XSLT 1.0: два элемента заменяют около 15 строк программного кода. Что еще лучше, новый код намного проще и удобнее в сопровождении.

Элемент <xsl:for-each-group> сам находит все уникальные значения <state>, и нам не приходится искать их самостоятельно. Но если бы вдруг потребовалось получить набор уникальных значений группировки, в XPath 2.0 and XQuery 1.0 имеется функция distinct-values(). Список уникальных значений в XSLT 2.0 строится примерно так:

```
<xsl:variable name="list-of-unique-states" as="xs:string*">
  <xsl:perform-sort>
    <xsl:sort select="states:getStateName(.)"/>
    <xsl:sequence
      select="distinct-values(document(/report/po/@filename)
              /purchase-order/customer/address/state)"/>
  </xsl:perform-sort>
</xsl:variable>
```

Мы создаем переменную типа xs:string*. Это означает, что переменная представляет собой последовательность строк, каждая из которых содержит уникальное сокращение штата. Затем новый элемент <xsl:perform-sort> сортирует уникальные значения по названию штата (не по сокращению). Внутри <xsl:perform-sort> находится элемент <xsl:sort>, который сортирует все значения по названию штата, а также элемент <xsl:sequence>, который выбирает все уникальные значения всех элементов <state>. (При этом используется функция states:getStateName().)

Хорошо, что в XSLT 2.0 существует это мощное и притом простое решение. Однако еще лучше, что <xsl:for-each-group> упрощает работу настолько, что даже оно становится лишним.

Вычисления без рекурсии

В исходной таблице стилей для вычисления суммы по каждому заказу использовалась рекурсия. Все элементы <item> передавались рекурсивному шаблону, который возвращал общую сумму заказа после обработки всех элементов <item>. В XSLT 2.0 эта задача решается простым выражением XPath. Всего одна строка кода:

```
<xsl:value-of
  select="format-number(sum(items/item/(qty * price)),
          '$#,###.00')"/>
```

Вот и все! Благодаря поддержке математических вычислений в выражениях XPath 2.0 задача становится до смешного тривиальной. Всего одна строка кода заменяет примерно 20 строк в исходной таблице.

Реализация таблиц отображения с помощью <xsl:function>

Реализация таблицы отображения на базе `document('')` работает, но функция XSLT позволяет упростить и этот аспект таблицы стилей. Вызов `getStateName()` преобразует сокращение штата в его название; он выглядит намного проще, чем вызов `document('')/*/states:name[@abbrev=current()]` или что-нибудь в этом роде. Функция выглядит так:

```
<xsl:function name="states:getStateName" as="xs:string">
  <xsl:param name="abbr" as="xs:string"/>
  <xsl:variable name="abbreviations" as="xs:string*"
    select="'AL', 'AK', 'AS', 'AZ', 'AR', 'CA', 'CO', 'CT',
<!-- Список сокращен для экономии места -->
            'WV', 'WI', 'WY'"/>
  <xsl:variable name="stateNames" as="xs:string*"
    select="'Alabama', 'Alaska', 'American Samoa','Arizona',
<!-- Список названий штатов сокращен для экономии места -->
            'West Virginia', 'Wisconsin', 'Wyoming'"/>
  <xsl:variable name="index"
    select="if (count(index-of($abbreviations, $abbr)) gt 0)
            then subsequence(index-of($abbreviations, $abbr), 1, 1)
            else 0"/>
  <xsl:value-of
    select="if ($index gt 0)
            then string(subsequence($stateNames, $index, 1))
            else ''"/>
</xsl:function>
```

Функция получает строковый параметр с сокращением штата и возвращает строку. В ней используются две переменные для хранения данных: `$abbreviations` – последовательность из всех сокращений, а `$stateNames` – последовательность из всех названий штатов. Между объектами этих двух последовательностей существует попарное соответствие. Если сокращение, переданное функции, соответствует пятому объекту последовательности `$abbreviations`, то название штата берется из пятого объекта последовательности `$stateNames`.

Позиция передаваемого сокращения вычисляется функцией XPath 2.0 and XQuery 1.0 `index-of()`. Функция возвращает последовательность значений `xs:integer`, каждое из которых представляет индекс совпадения. Функция `count()` подсчитывает количество объектов в последовательности, возвращаемой `index-of()`. Если сокращение, переданное функции, совпадает хотя бы с одним объектом из последовательности `$abbreviations`, размер последовательности результата будет больше нуля (впрочем, в нашем примере он равен либо 1, либо 0). Чтобы узнать индекс совпавшего объекта, мы вызываем новую функцию `subsequence()` для получения первого объекта в последовательности. Если совпадения не найдены, индекс равен нулю.

Вычислив индекс совпавшего сокращения, функция возвращает соответствующее название штата. Если значение `$index` больше нуля, функция `subsequence()` возвращает соответствующий объект из последовательности `$stateNames`. Если значение `$index` равно нулю (переданное сокращение отсутствует в списке), функция возвращает пустую строку.

Обратите внимание на использование оператора XPath 2.0 and XQuery 1.0 `if` для упрощения кода. Того же эффекта можно было добиться при помощи `<xsl:choose>`, `<xsl:when>` и `<xsl:otherwise>`, но вариант с `if` гораздо проще. Обратите также внимание на оператор сравнения `gt`; в данном случае мы можем использовать его, потому что сравниваются два атомарных значения. Можно было бы использовать `>` или `&gt` – с теми же результатами.

Итак, функция успешно определена, и вызов `states:getStateName('AL')` возвращает строку `Alabama`. Теперь этот заметно упрощенный синтаксис можно использовать всюду, где потребуется. Вся логика преобразования сокращений изолируется в одном месте таблицы стилей, и нам уже не приходится использовать `document('')` с соответствующим выражением XPath каждый раз, когда потребуется получить название штата.

Использование if вместо <xsl:choose>

Хотя элементы XSLT `<xsl:if>`, `<xsl:choose>`, `<xsl:when>` и `<xsl:otherwise>` по своим возможностям аналогичны конструкции `if-then-else` в большинстве языков программирования, запись с этими элементами получается довольно громоздкой. Для упрощения кода стоит воспользоваться оператором XPath 2.0 and XQuery 1.0 `if`.

В таблице стилей XSLT 1.0 элемент `<xsl:choose>` используется в двух местах: для выбора фонового цвета строк таблицы и для изменения числового формата денежных сумм в первой строке каждого заказа. Первый элемент `<xsl:choose>`:

```
<tr>
  <xsl:attribute name="style">
    <xsl:text>background: </xsl:text>
    <xsl:choose>
      <xsl:when test="position() mod 2">
        <xsl:text>#CCCCFF</xsl:text>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>#66FF66</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:text>;</xsl:text>
  </xsl:attribute>
```

Мы создаем атрибут с именем style. Если строка имеет нечетный номер (результат position() mod 2 равен 1, что интерпретируется как true), выбирается цвет фона #CCCCFF; в противном случае назначается цвет #66FF66. В таблице стилей XSLT 2.0 эти элементы заменяются следующим шаблоном значения атрибута:

```
<tr style="{if (position() mod 2)
            then 'background: #CCCCFF;'
            else 'background: #66FF66;'}">
```

Второе применение <xsl:choose> (с выбором числового формата) в XSLT 2.0 реализуется так:

```
<xsl:value-of
  select="if (position() = 1)
          then format-number(price * qty, '$#,###.00')
          else format-number(price * qty, '#,###.00')"/>
```

В обоих случаях несколько элементов заменяются одной командой if внутри атрибута.

Функция format-date()

Осталось внести последнее усовершенствование: для форматирования даты заказа будет использоваться функция format-date(). Впрочем, здесь речь идет о расширении таблицы стилей XSLT 1.0; в предыдущих примерах функции XSLT 1.0 заменялись эквивалентными, но намного более простыми решениями. Дата оформления заказа хранится в трех атрибутах year, month и day. Мы воспользуемся значениями трех атрибутов для построения нового значения типа xs:date, а затем отформатируем его функцией format-date().

Для создания нового значения xs:date необходима строка в формате гггг-мм-дд. Например, выражение xs:date('2006-10-10') является допустимым вызовом конструктора xs:date. Однако ситуация осложняется тем, что значения месяца и дня *должны* состоять из двух цифр. Иначе говоря, вызов xs:date('2006-9-8') приводит к ошибке. Однако в наших заказах значения месяца и дня могут состоять и из одной цифры, поэтому нам придется написать код для добавления начального нуля, если соответствующее значение меньше 10. Вот как это делается:

```
<xsl:variable name="monthValue" as="xs:string"
  select="if (date/@month &lt; 10)
          then concat('0', date/@month)
          else date/@month"/>
<xsl:variable name="dayValue" as="xs:string"
  select="if (date/@day &lt; 10)
          then concat('0', date/@day)
          else date/@day"/>
```

Значения атрибутов сравниваются с 10; если атрибут меньше, к значению добавляется строка '0'. (Обратите внимание: обе создаваемые переменные относятся к типу xs:string.) В этом коде нам приходится использовать оператор <, потому что узел сравнивается с числом. Чтобы использовать оператор lt, нам пришлось бы выбрать запись if (number(date/@day) lt 10).

После того как месяц и день будут нормализованы, мы можем создать значение xs:date и отформатировать его. Оставшаяся часть кода выглядит так:

```
<xsl:variable name="orderDate" as="xs:date"
  select="xs:date(concat(date/@year, '-',
          $monthValue, '-', $dayValue))"/>
<xsl:value-of
  select="format-date($orderDate, '[FNn], [MNn] [D1], [Y]')"/>
```

В нашем примере дата 2006-10-10 форматируется в виде Tuesday, October 10, 2006. Чтобы сделать то же самое в XSLT 1.0, необходимо создать таблицу отображения для преобразования номеров месяцев в их текстовые эквиваленты. Кроме того, понадобится таблица отображения для названий дней недели, однако придется писать функцию расширения, чтобы определить, что 10 октября 2006 года было вторником.

Полное решение для XSLT 2.0

А вот как выглядит итоговая таблица стилей в XSLT 2.0:

```
<?xml version="1.0"?>
<!-- masterdox5.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:states="http://www.usps.com/ncsc/lookups/abbreviations.html"
  exclude-result-prefixes="xs states">

  <xsl:variable name="purchase-orders"
    select="document(/report/po/@filename)/purchase-order"/>

  <xsl:function name="states:getStateName" as="xs:string">
    <xsl:param name="abbr" as="xs:string"/>
    <xsl:variable name="abbreviations" as="xs:string*"
      select="'AL', 'AK', 'AS', 'AZ', 'AR', 'CA', 'CO', 'CT',
<!-- Список сокращен для экономии места -->
              'WV', 'WI', 'WY'"/>
    <xsl:variable name="stateNames" as="xs:string*"
      select="'Alabama', 'Alaska', 'American Samoa','Arizona',
<!-- Список названий штатов сокращен для экономии места -->
              'West Virginia', 'Wisconsin', 'Wyoming'"/>
    <xsl:variable name="index"
      select="if (count(index-of($abbreviations, $abbr)) gt 0)
              then subsequence(index-of($abbreviations, $abbr), 1, 1)
              else 0"/>
```

```
  <xsl:value-of
    select="if ($index gt 0)
            then string(subsequence($stateNames, $index, 1))
            else ''"/>
</xsl:function>

<xsl:output method="html"/>

<xsl:template match="/">
  <html>
    <head>
      <title><xsl:value-of select="/report/title"/></title>
    </head>
    <body style="font-family: sans-serif;">
      <h1>
        Selected Purchase Orders -
        <span style="font-style: italic;
                     text-decoration: underline;">Grouped</span>
        by state name
      </h1>
      <xsl:for-each-group
        select="$purchase-orders" group-by="customer/address/state">
        <xsl:sort
          select="states:getStateName(current-grouping-key())"/>
        <h2 style="color: white; background: black;">
          Purchase Orders from
          <xsl:value-of
            select="states:getStateName(current-grouping-key())"/>
        </h2>
        <xsl:for-each select="current-group()">
          <xsl:sort select="customer/address/city"/>
          <xsl:apply-templates select="."/>
        </xsl:for-each>
      </xsl:for-each-group>
    </body>
  </html>
</xsl:template>

<xsl:template match="purchase-order">
  <h3>
    <xsl:value-of
      select="customer/address[@type='business']/name/title"/>
    <xsl:text> </xsl:text>
    <xsl:value-of
      select="customer/address[@type='business']/name/first-name"/>
    <xsl:text> </xsl:text>
    <xsl:value-of
      select="customer/address[@type='business']/name/last-name"/>
    <xsl:text> - </xsl:text>
    <xsl:value-of
      select="customer/address[@type='business']/city"/>
    <xsl:text>, </xsl:text>
```

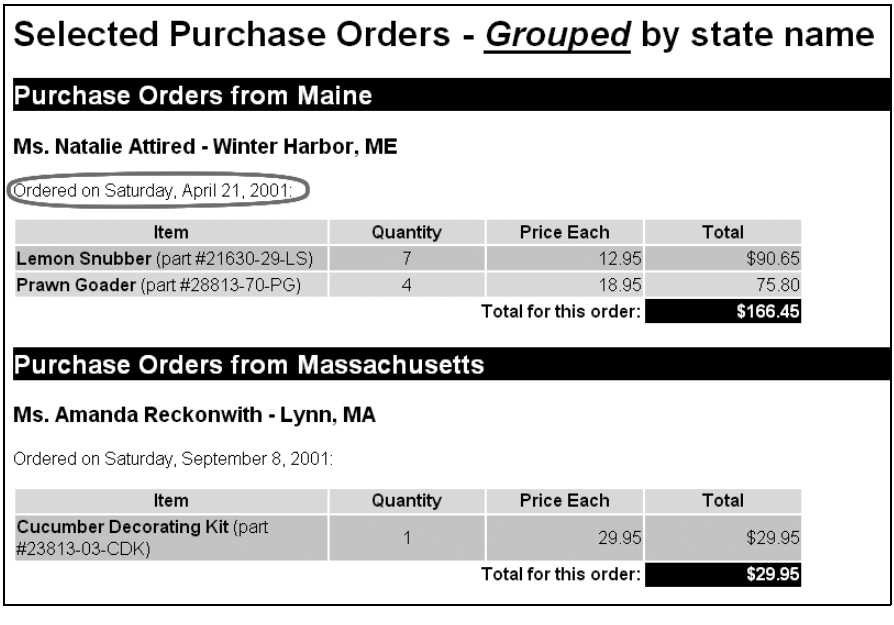
```
  <span style="font-weight: bold;">
    <xsl:value-of
      select="customer/address[@type='business']/state"/>
  </span>
</h3>
<p>
  <xsl:text>Ordered on </xsl:text>
  <xsl:variable name="monthValue" as="xs:string"
    select="if (date/@month &lt; 10)
            then concat('0', date/@month)
            else date/@month"/>
  <xsl:variable name="dayValue" as="xs:string"
    select="if (date/@day &lt; 10)
            then concat('0', date/@day)
            else date/@day"/>
  <xsl:variable name="orderDate" as="xs:date"
    select="xs:date(concat(date/@year, '-',
                           $monthValue, '-', $dayValue))"/>
  <xsl:value-of
    select="format-date($orderDate, '[FNn], [MNn] [D1], [Y]')"/>
  <xsl:text>:</xsl:text>
</p>
<table width="80%" border="0">
  <tr style="background: #66FF66;">
    <th>Item</th>
    <th>Quantity</th>
    <th>Price Each</th>
    <th>Total</th>
  </tr>
  <xsl:for-each select="items/item">
    <tr style="{if ((position() mod 2) = 1)
                then 'background: #CCCCFF;'
                else 'background: #66FF66;'}">
      <td width="40%">
        <span style="font-weight: bold;">
          <xsl:value-of select="name"/>
        </span>
        <xsl:text> (part #</xsl:text>
        <xsl:value-of select="@part-no"/>
        <xsl:text>)</xsl:text>
      </td>
      <td style="text-align: center;" width="20%">
        <xsl:value-of select="qty"/>
      </td>
      <td style="text-align: right;" width="20%">
        <xsl:value-of select="price"/>
      </td>
      <td style="text-align: right;" width="20%">
        <xsl:value-of
          select="if (position() = 1)
                  then format-number(price * qty, '$#,###.00')
                  else format-number(price * qty, '#,###.00')"/>
```

```
          </td>
        </tr>
      </xsl:for-each>
      <tr style="font-weight: bold;">
        <td colspan="3" style="text-align: right;">
          Total for this order:
        </td>
        <td align="right" style="color: white; background: black;">
          <xsl:value-of
            select="format-number(sum(items/item/(qty * price)),
                                  '$#,###.00')"/>
        </td>
      </tr>
    </table>
  </xsl:template>

</xsl:stylesheet>
```

Результат применения таблицы стилей показан на рис. 8.5.

Форматирование дат в приведенной таблице отличается от версии XSLT 1.0; в остальном вывод идентичен, хотя новая таблица стилей намного короче.



Selected Purchase Orders - *Grouped* by state name

Purchase Orders from Maine

Ms. Natalie Attired - Winter Harbor, ME

Ordered on Saturday, April 21, 2001:

Item	Quantity	Price Each	Total
Lemon Snubber (part #21630-29-LS)	7	12.95	$90.65
Prawn Goader (part #28813-70-PG)	4	18.95	75.80
		Total for this order:	**$166.45**

Purchase Orders from Massachusetts

Ms. Amanda Reckonwith - Lynn, MA

Ordered on Saturday, September 8, 2001:

Item	Quantity	Price Each	Total
Cucumber Decorating Kit (part #23813-03-CDK)	1	29.95	$29.95
		Total for this order:	**$29.95**

Рис. 8.5. *Заказы, сгруппированные по названию штата (версия для XSLT 2.0)*

[2.0] Функции doc() и doc-available()

В XSLT 2.0 появилась новая функция doc(), которая очень похожа на функцию document(), но в некоторых отношениях проще ее. Прежде всего, функция document() получает в первом аргументе набор узлов или последовательность, а функция doc() получает одну строку. В предыдущих примерах мы создавали переменную, содержащую все узлы из всех документов, ссылки на которые присутствовали в списке заказов:

```
<xsl:variable name="purchase-orders"
  select="document(/report/po/@filename)/purchase-order"/>
```

Элемент возвращает последовательность узлов purchase-order, каждый из которых имеет структуру, определенную в заказе. Переменная $purchase-orders может использоваться в элементах <xsl:for-each> и <xsl:for-each-group>. Если исходный документ содержит более одного элемента <po>, таблица стилей со следующей разметкой выдает ошибку:

```
<!-- Не работает с документом из нашего примера -->
<xsl:variable name="purchase-orders"
  select="doc(/report/po/@filename)/purchase-order"/>
```

Если мы хотим использовать функцию doc(), возможно несколько вариантов. Можно указать конкретный элемент <po>:

```
<xsl:variable name="purchase-orders"
  select="doc(/report/po[2]/@filename)/purchase-order"/>
```

Более практичное решение основано на включении doc() в элемент <xsl:for-each>:

```
<xsl:for-each select="/report/po">
  <xsl:variable name="purchase-orders"
    select="doc(@filename)/purchase-order"/>
</xsl:for-each>
```

Кроме того, можно изменить входной документ так, чтобы он содержал только один элемент <po>; такое решение вряд ли можно назвать практичным, но функция doc() будет работать.

Вторым различием между doc() и document() является то, что функция doc() возвращает узел документа одного документа XML, тогда как функция document() может возвращать набор узлов из нескольких документов. В большинстве случаев стоит использовать более гибкую функцию document().

Ранее уже говорилось о том, что функция document() имеет необязательный второй аргумент, задающий базовый URI-адрес для разрешения относительных URL. *У функции* doc() *такая возможность отсутствует*; базовый URI-адрес всегда совпадает с базовым URI-адресом таблицы стилей.

Последнее различие между doc() и document() заключается в том, что функция document() поддерживает идентификаторы фрагментов в URL, а функция doc() их *не поддерживает*. Если ваш процессор использует идентификаторы фрагментов, функция document() может вернуть конкретные узлы в документе вместо того, чтобы просто вернуть узел документа.

```
<!-- Это возможно: -->
<xsl:variable name="mainContent"
  select="document('http://www.ibm.com/developerworks/index.html#main')"/>

<!-- А это нет: -->
<xsl:variable name="mainContent"
  select="doc('http://www.ibm.com/developerworks/index.html#main')"/>
```

Небольшой фрагмент первой таблицы стилей (masterdox1.xsl), переписанной для использования doc() вместо document():

```
<?xml version="1.0"?>
<!-- doc.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of select="/report/title"/></title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>Selected Purchase Orders - Unsorted</h1>
        <xsl:for-each select="/report/po">
          <xsl:apply-templates
            select="doc(@filename)/purchase-order"/>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
...
```

Таблица стилей структурирована таким образом, что каждый атрибут filename передается функции doc() в виде строки. Таблица выдает тот же результат, что и исходная таблица стилей; впрочем, переписать более сложные таблицы стилей этой главы так, чтобы в них использовалась функция doc() вместо document(), будет не очень просто.

В дополнение к функции doc() спецификация XPath 2.0 and XQuery 1.0 предоставляет функцию doc-available(). Функция возвращает true, если заданный URL-адрес доступен; в противном случае возвращается false. Пример простой таблицы стилей с использованием функции doc-available():

```
<?xml version="1.0"?>
<!-- doc-available.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the doc-available() function:&#xA;</xsl:text>

    <xsl:text>&#xA;  doc-available('polist.xml') = </xsl:text>
    <xsl:value-of select="doc-available('polist.xml')"/>

    <xsl:text>&#xA;&#xA;  doc-available('polist2.xml') = </xsl:text>
    <xsl:value-of select="doc-available('polist2.xml')"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Tests of the doc-available() function:

  doc-available('polist.xml') = true

  doc-available('polist2.xml') = false
```

Функция `doc-available()` позволяет заранее проверить, не произойдет ли ошибка при открытии конкретного URL-адреса. Если `doc-available()` возвращает `true`, открытие URL-адреса функцией `doc()` безопасно.

Для аргумента `doc-available()` устанавливаются те же ограничения, что и для аргумента `doc()`. Если `doc-available()` передается аргумент, недействительный для `doc()` (например, упоминавшееся ранее выражение `doc(/report/po/@filename)/purchase-order`), функция `doc-available()` возвращает `false`.

Итак, если `doc-available()` возвращает `true`, функция `document()` будет работать. С другой стороны, даже если `doc-available()` возвращает `false`, может оказаться, что `document()` все равно будет работать. В XSLT 2.0 не существует аналогичной функции `document-available()`.

[2.0] Функция collection()

Функция `collection()` получает строковый аргумент и возвращает коллекцию узлов. Она определяется в спецификации XPath 2.0 и предоставляет возможность использования URI-адреса для выборки коллекции документов. Способ хранения документов (и вообще их интерпретация как документов) зависит от реализации. В частности, спецификация упоминает выборку из реляционной базы данных как одну из возможных реализаций функции. Возможность динамического построения передаваемой функции строки с возможным включением параметров делает функцию `collection()` очень гибкой.

Рассмотрим короткий пример. Функции `collection()` передается следующий документ:

```
<?xml version="1.0"?>
<!-- polist.xml -->
<collection>
  <doc href="po38292.xml"/>
  <doc href="po38293.xml"/>
  <doc href="po38294.xml"/>
  <doc href="po38295.xml"/>
</collection>
```

Этот документ очень похож на список заказов, с которым мы работали ранее в этой главе. Таблица стилей, в которой вызывается функция `collection()`, выглядит так:

```
<?xml version="1.0"?>
<!-- collection.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the collection() function:</xsl:text>

    <xsl:variable name="docPile" as="node()*"
      select="collection('polist.xml')"/>

    <xsl:text>&#xA;&#xA;  The customers in the </xsl:text>
    <xsl:text>collection are: &#xA;    </xsl:text>
    <xsl:for-each select="$docPile/purchase-order/customer">
      <xsl:sort select="address/name/last-name"/>
      <xsl:value-of
        select="address/name/title,
                address/name/first-name,
                address/name/last-name"
        separator=" "/>
      <xsl:text>    &#xA;    </xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей получает узлы от функции `collection()` и сохраняет их в переменной `$docPile`. Получив узлы, мы сортируем всех клиентов в заказах по фамилии, а затем записываем их в выходной документ. Результат выглядит так:

```
A test of the collection() function:

  The customers in the collection are:
    Ms. Natalie Attired
    Mrs. Mary Backstayge
    Mr. Chester Hasbrouck Frisby
    Ms. Amanda Reckonwith
```

За подробностями обращайтесь к описанию функции *[2.0]* `collection()` в приложении C.

[2.0] Функции unparsed-text() и unparsed-text-available()

Последняя новая функция объединения документов – `unparsed-text()` – предназначена для чтения текста по URL-адресу. Текст не подвергается синтаксическому разбору, что позволяет использовать функцию для чтения текстовых документов, разделенных запятыми данных и даже документов HTML, не содержащих корректного (well-formed) кода XML. Более того, `unparsed-text()` может использоваться в сочетании с такими новыми возможностями, как функция `tokenize()` или элемент `<xsl:analyze-string>`, для обработки текста и его полезных преобразований.

В следующем примере мы прочитаем файл с данными, разделенными запятыми, и выведем их в таблице HTML. Исходный файл *unparsed-text.csv* выглядит так:

```
Mr.,Chester Hasbrouck,Frisby,1234 Main Street,Sheboygan,WI,48392
Ms.,Natalie,Attired,707 Breitling Way,Winter Harbor,ME,00218
Ms.,Amanda,Reckonwith,930-A Chestnut Street,Lynn,MA,02930
Mrs.,Mary,Backstayge,283 First Avenue,Skunk Haven,MA,02718
```

Обработка данных состоит из трех простых этапов. На первом этапе функция `tokenize()` читает очередную строку из файла. Затем функция `tokenize()` получает все значения, разделенные запятыми. На последнем этапе каждое значение преобразуется нужным образом. В представленном здесь файле с данными, разделенными запятыми, третье значение в каждой строке содержит фамилию клиента, седьмое значение содержит почтовый индекс и т. д.

Для построчной обработки файла мы воспользуемся готовым решением, приведенным в спецификации XSLT 2.0:

```
<xsl:for-each
  select="tokenize(unparsed-text('addresses.csv'), '\r?\n')">
```

Элемент `<xsl:for-each>` последовательно обрабатывает каждую строку файла, а регулярное выражение `\r?\n` совпадает с концом строки. (Как указано в спецификации, `unparsed-text()` не нормализует завершения строк, поэтому мы должны предусмотреть необязательный символ возврата курсора `\r?`.)

В цикле `<xsl:for-each>` строка разбивается на лексемы. Задача решается легко: нас интересуют значения, находящиеся между запятыми. Функция `tokenize()` возвращает последовательность, которая сохраняется в переменной:

```
<xsl:variable name="tokens" select="tokenize(., ',')"/>
```

Прежде чем обрабатывать значения, разделенные запятыми, мы при помощи функции `count()` убеждаемся в том, что в строке были обнаружены полезные данные. Если файл CSV содержит пустую строку, количество лексем в этой строке равно нулю; пустая строка игнорируется, и обработка продолжается со следующей строки. В оставшейся части таблицы для выборки нужных значений из последовательности используется функция `subsequence()`.

Полный код таблицы стилей, обрабатывающей CSV-файл:

```
<?xml version="1.0"?>
<!-- unparsed-text.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Customer Addresses</title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>Customer Addresses</h1>
        <table width="60%" border="0">
          <tr style="background: #66FF66;">
            <th>Name</th>
            <th>Street</th>
            <th>City</th>
            <th>State</th>
            <th>Zip</th>
          </tr>
          <xsl:for-each
            select="tokenize(unparsed-text('addresses.csv'), '\r?\n')">
            <xsl:variable name="tokens" select="tokenize(., ',')"/>
            <xsl:if test="count($tokens)">
              <tr style="{if (position() mod 2)
                          then 'background: #CCCCFF;'
                          else 'background: #66FF66;'}">
                <td width="40%">
                  <xsl:value-of
                    select="subsequence($tokens, 1, 2)" separator=" "/>
                  <xsl:text> </xsl:text>
                  <span style="font-weight: bold; font-size: 125%;">
                    <xsl:value-of select="subsequence($tokens, 3, 1)"/>
                  </span>
                </td>
                <td width="20%">
                  <xsl:value-of select="subsequence($tokens, 4, 1)"/>
                </td>
                <td width="20%">
                  <xsl:value-of select="subsequence($tokens, 5, 1)"/>
```

```
            </td>
            <td width="10%" style="text-align: center;">
              <xsl:value-of select="subsequence($tokens, 6, 1)"/>
            </td>
            <td width="10%" style="text-align: center;">
              <xsl:value-of select="subsequence($tokens, 7, 1)"/>
            </td>
          </tr>
        </xsl:if>
      </xsl:for-each>
    </table>
  </body>
 </html>
</xsl:template>

</xsl:stylesheet>
```

Результат показан на рис. 8.6.

Customer Addresses

Name	Street	City	State	Zip
Mr. Chester Hasbrouck **Frisby**	1234 Main Street	Sheboygan	WI	48392
Ms. Natalie **Attired**	707 Breitling Way	Winter Harbor	ME	00218
Ms. Amanda **Reckonwith**	930-A Chestnut Street	Lynn	MA	02930
Mrs. Mary **Backstayge**	283 First Avenue	Skunk Haven	MA	02718

Рис. 8.6. *Документ HTML, сгенерированный на основе данных, разделенных запятыми*

Обратите внимание: первые две лексемы (приставка и имя клиента) выбираются в виде последовательности, а затем записываются в выходной документ с атрибутом `separator=" "`. Если приставка отсутствует, таблица стилей не вставляет лишний пробел перед именем клиента.

Функция `unparsed-text()` может получать необязательный второй аргумент, который определяет кодировку документа, находящегося по URL-адресу. Например, если файл *addresses.csv* сохранен в кодировке UTF-16, чтение файла может осуществляться вызовом `unparsed-text('addresses.csv', 'utf-16')`.

Функция `unparsed-text()` может также использоваться для импортирования файлов XML и HTML без синтаксического разбора. За дополнительными примерами обращайтесь к описанию функции `unparsed-text()`.

Новая функция `unparsed-text-available()` является аналогом `doc-available()`. Она возвращает `true`, если неразобранный текст доступен по заданному URL-адресу; в противном случае возвращается `false`.

Итоги

В главе представлен обзор функций XSLT и XPath, работающих с несколькими документами. Эти мощные функции позволяют генерировать выходные документы, содержащие элементы из нескольких исходных документов. В рассмотренных примерах мы сгенерировали несколько представлений входных документов, однако существует много других полезных возможностей. Главное преимущество этих функций заключается в том, что они позволяют определять сводные представления нескольких документов, отделенные от самих документов. Если потребуется определить новое представление, входные документы останутся неизменными.

Также была рассмотрена новая функция `unparsed-text()`, предназначенная для чтения данных, не хранящихся в формате XML, и их включения в итоговые документы. Функции этой главы обеспечивают немалую экономию времени и усилий разработчика при построении отчетов и других сводных документов.

9

Расширение XSTL

К настоящему моменту мы потратили немало времени, разбираясь в том, как применять встроенные возможности XSLT и XPath для решения поставленных задач. Мы также поговорили о несколько необычной модели обработки данных, которая может вызывать затруднения у программистов с опытом работы на процедурных языках. Но что делать, если стандартных возможностей XSLT и XPath оказывается недостаточно?

В этой главе рассматривается механизм расширения XSLT, который позволяет дополнять язык новыми функциями и элементами. Стандарт XSLT не определяет механизм расширения во всех подробностях, поэтому между процессорами существуют некоторые различия. Конечно, если вы написали функцию расширения или элемент для своего любимого процессора, разработчики других процессоров не смогут сделать ничего, что бы помешало работе ваших функций или элементов. Но если вы вдруг решите сменить процессор XSLT, скорее всего, код расширения придется изменять.

Заодно мы также обсудим проект EXSLT, направленный на создание общей библиотеки функций расширения, работающих в разных процессорах XSLT.

Примеры этой главы написаны для процессоров Xalan и Saxon на базе Java, а также для .NET Framework (с использованием C#). Мы обсудим, как пишутся таблицы стилей, работающие в разных процессорах, а также кратко рассмотрим различия между API, поддерживаемыми этими процессорами. К тому же процессор Xalan-J поддерживает инфраструктуру Apache Bean Scripting Framework (BSF); это означает, что расширения можно писать на языках Jython (также известен как JPython), JavaScript, Jacl и любых других, поддерживаемых BSF.

Механизм расширения XSLT

Стандарт XSLT определяет два вида расширений: элементы расширения и функции расширения. В спецификации также определяется механизм *отката* (*fallback processing*) – корректной реакции на недоступность элементов и функций расширения. (Откат применяется и в том случае, когда процессор XSLT 1.0 должен обработать таблицу стилей XSLT 2.0.) Сначала мы в общих чертах рассмотрим эти возможности, а затем перейдем к примерам, в полной мере демонстрирующим расширения и откат.

Помните, что спецификация XSLT определяет лишь то, как должен работать механизм расширения; она *не определяет* его реализацию. Как будет показано в этой главе, процессоры XSLT реализуют расширения по-разному. Например, процессор XSLT на базе Java может реализовать функцию или элемент расширения в виде класса Java, а процессор XSLT на базе .NET использует класс .NET. Как эти классы назначаются, зависит от конкретного процессора.

Элементы расширения

Элементом расширения называется элемент, который должен обрабатываться блоком кода, внешним по отношению к процессору XSLT. Как и следовало ожидать, подробности реализации зависят от процессора. В этой главе мы обсудим, как элемент расширения может обращаться к представлению XPath исходного документа XML, как он обрабатывает шаблоны значений атрибутов (и возможно ли это), как он генерирует вывод и перемещается по дереву XPath для обработки исходного документа. Мы рассмотрим различные API для выполнения всех перечисленных операций; конечно, эти API существенно зависят от конкретного процессора. Наконец, хотя процессоры XSLT обычно предоставляют разработчику расширения доступ к исходному документу XML, стандарт не определяет набор функций или методов доступа, которые должны поддерживаться.

Функции расширения

Как нетрудно догадаться, функция расширения определяется в блоке кода, вызываемом из таблицы стилей XSLT. В некоторых случаях эта функция пишется на сценарном языке в самой таблице стилей. В таких случаях сценарный код содержится в элементе XML, не принадлежащем пространству имен XSLT. В других случаях функция определяется в отдельном файле, загружаемом процессором XSLT во время выполнения. Код функции расширения пишется на сценарном или на компилируемом языке.

Независимо от способа записи или обращения функции расширения могут передаваться параметры, и функция может возвращать результат. Этот результат может относиться к любому из типов данных, под-

держиваемых XPath. Вдобавок процессоры XSLT могут позволять функциям расширения возвращать другие типы данных, хотя последние должны обрабатываться другими функциями, возвращающими один из типов данных XPath.

Откат

Последней частью механизма расширения XSLT является элемент <xsl:fallback>. Этот элемент обрабатывается в том случае, когда его внешний элемент не поддерживается процессором XSLT. Например, элемент <xsl:result-document> появился только в XSLT 2.0. Мы можем поместить элемент <xsl:fallback> внутри элемента <xsl:result-document>. Таблица стилей передается процессору XSLT 1.0, который не поддерживает элемент <xsl:result-document>. Процессор XSLT 1.0 просматривает элемент <xsl:result-document> в поисках элемента <xsl:fallback>; обнаружив такой элемент, он обрабатывает его содержимое. Как будет показано позднее в этой главе, сам элемент <xsl:fallback> тоже может содержать внутренний элемент <xsl:fallback>. Элемент <xsl:fallback> дает процессору XSLT возможность корректно выйти из положения, если он не может выполнить нужную задачу.

Пространства имен расширений

Элементы и функции расширения XSLT должны обладать префиксом пространства имен, отличным от префикса пространства имен XSLT. Например, таблица стилей, объявляющая несколько префиксов пространств имен, может начинаться так:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:xalan-java="http://xml.apache.org/xslt/java"
  xmlns:saxon-java="java:java.lang.Math"
  extension-element-prefixes="xalan-java saxon-java">
```

По умолчанию процессор XSLT копирует все пространства имен, не связанные с XSLT, в выходной документ. Это означает, что пространства имен svg, xalan-java и saxon-java будут определены в выходном документе. В приведенном примере документ SVG генерируется с использованием расширений, связанных с пространствами имен xalan-java и saxon-java. Однако нам не нужно, чтобы пространства имен расширений определялись в выходном документе, поэтому мы предотвращаем их перенесение в сгенерированный документ атрибутом extension-element-prefixes. С другой стороны, пространство имен svg присутствует в выходном документе, а это именно то, что требуется.

[2.0] Создание новых функций с помощью элемента <xsl:function>

В XSLT 2.0 появился новый элемент `<xsl:function>`, позволяющий определять пользовательские функции прямо в таблице стилей. Это самая простая разновидность расширений, рассматриваемая в этой главе; расширение определяется в том же файле и с тем же синтаксисом, что и таблица стилей, а стандарт очень четко определяет все аспекты определения и активизации функции. В нашем примере простая таблица стилей строит таблицу HTML, ячейкам которой поочередно назначаются четыре цвета фона. По заданной текущей позиции наша функция возвращает одно из четырех значений.

Чтобы определить функцию, необходимо выполнить несколько действий: определить пространство имен для функции (отличное от пространства имен XSLT), задать имя функции, определить возвращаемый тип данных, а также определить имена и типы данных всех параметров функции. Пространство имен определяется достаточно просто; не забудьте включить префикс пространства имен в атрибут `exclude-result-prefixes` элемента `<xsl:stylesheet>`. Наша функция называется `getBackgroundColor` и возвращает значение `xs:string` с именем фонового цвета каждой ячейки таблицы. Наконец, при вызове функции передается значение `xs:integer` с текущей позицией. Функция выглядит так:

```
<xsl:function name="sample:getBackgroundColor" as="xs:string">
  <xsl:param name="pos" as="xs:integer"/>
  <xsl:value-of select="$colors[($pos mod count($colors)) + 1]"/>
</xsl:function>
```

Функция принадлежит пространству имен `sample`, получает единственный целочисленный параметр и возвращает строку. Названия цветов берутся из переменной `$colors`. Вместо того чтобы брать данные из документа XML, мы создадим последовательность чисел и поместим каждое из них в ячейку таблицы. Таблица стилей выглядит так:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- simple-function.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:sample="http://www.oreilly.com/catalog/xslt"
  exclude-result-prefixes="xs sample">

  <xsl:output method="html" include-content-type="no"/>

  <xsl:variable name="colors" as="xs:string *"
    select="('green', 'grey', 'blue', 'red')"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>A table with different background colors</title>
```

```
      </head>
      <body style="font-family: sans-serif;">
        <h1 style="font-size: 28;">
          A table with different background colors:
        </h1>
        <table border="3" cellpadding="5" cellspacing="5" width="50%">
          <tr>
            <xsl:for-each select="1 to 12">
              <td style="font-size: 48; color: black;
                         font-weight: bold; text-align: center;"
                bgcolor="{sample:getBackgroundColor(position())}">
                <xsl:value-of select="."/>
              </td>
            </xsl:for-each>
          </tr>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:function name="sample:getBackgroundColor" as="xs:string">
    <xsl:param name="pos" as="xs:integer"/>
    <xsl:value-of select="$colors[($pos mod count($colors)) + 1]"/>
  </xsl:function>

</xsl:stylesheet>
```

Оператор XPath 2.0 `to` строит последовательность значений, после чего функция `getBackgroundColor` задает цвет фона каждой ячейки таблицы. На рис. 9.1 показан вид сгенерированного документа в броузере.

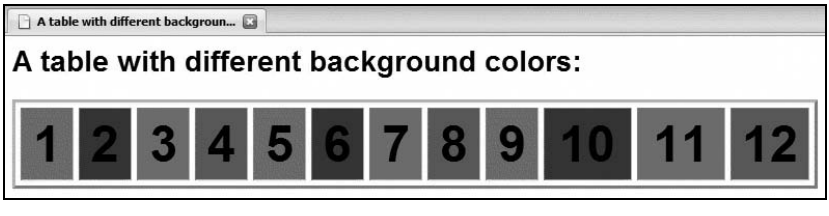


***Рис. 9.1.** Документ, сгенерированный с помощью элемента <xsl:function>*

Чтобы реализовать эквивалентную функцию в XSLT 1.0, нам пришлось бы использовать логику следующего вида:

```
<xsl:choose>
  <xsl:when test="position() mod 4 = 0">
    <!-- для цвета фона xxx -->
  </xsl:when>
  <xsl:when test="position() mod 4 = 1">
    <!-- для цвета фона xxx -->
  </xsl:when>
  ...
</xsl:choose>
```

Такой код занимает больше места, и его труднее сопровождать. Скажем, при изменении количества цветов в последовательности придется изменять код XSLT 1.0. Напротив, функция XSLT 2.0 определяет длину последовательности в своих вычислениях, поэтому список можно дополнить любым количеством цветов без изменения кода функции.

Пример: построение нескольких выходных файлов

Вся суть расширений заключается в наделении процессора XSLT новыми возможностями. К числу самых распространенных задач такого рода относится возможность построения нескольких выходных документов. Как было показано ранее, функция `document()` позволяет работать с несколькими исходными документами, однако аналогичной возможности для выходных документов в XSLT 1.0 не существует. Поддержка XSLT 2.0 в Saxon включает элемент `<xsl:result-document>`, который позволяет сгенерировать несколько выходных документов. Процессор Xalan не поддерживает XSLT 2.0, но он поддерживает элемент расширения `<redirect:write>`, который делает то же самое. Мы рассмотрим таблицу стилей, в которой элемент `<xsl:fallback>` обеспечивает полезный результат независимо от процессора. При использовании Saxon или Xalan мы получим несколько выходных документов, связанных гиперссылками; в любом другом процессоре будет создан один файл HTML, содержащий ту же информацию. В примере используется следующий исходный документ:

```
<?xml version="1.0"?>
<!-- chapters.xml -->
<book>
  <title>XSLT Topics</title>
  <chapter>
    <title>XPath</title>
    <para>If this chapter had any text, it would appear here.</para>
  </chapter>
  <chapter>
    <title>Stylesheet Basics</title>
    <para>If this chapter had any text, it would appear here.</para>
  </chapter>
...
  <chapter>
    <title>Combining XML Documents</title>
    <para>If this chapter had any text, it would appear here.</para>
  </chapter>
</book>
```

Кроме элемента `<xsl:fallback>` в таблице стилей также используется функция `element-available()`, проверяющая доступность элемента. Если построить несколько выходных документов невозможно (т. е. если необходимые элементы недоступны), создается единый файл HTML.

Если элементы доступны, сначала создается файл HTML с гиперссылками на отдельные файлы HTML, которые создаются позднее в этой таблице стилей. При создании отдельных файлов используются элементы <xsl:result-document> и <redirect:write> с вставками <xsl:fallback> для процессоров XSLT, которые эти элементы не поддерживают.

Рассмотрим основные части этого примера. Сначала элемент <xsl:stylesheet> определяет префикс пространства имен redirect и сообщает ядру XSLT, что префикс будет использоваться для ссылок на элемент расширения:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- multiple-output-files.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:redirect="org.apache.xalan.xslt.extensions.Redirect"
  extension-element-prefixes="redirect">
```

Атрибут extension-element-prefixes сообщает процессору XSLT, что все элементы и функции с префиксом redirect должны обрабатываться расширением. Когда процессор обнаруживает что-либо, принадлежащее этому пространству имен, он загружает код расширения и приказывает ему обработать запрос от таблицы стилей. Если вы забудете определить перенаправление для префикса элемента расширения, Xalan-J выдаст ошибку.

Синтаксис всего, что происходило до настоящего момента, был достаточно стандартным, однако существуют довольно значительные различия в том, как процессоры XSLT поступают с определяемой информацией. Например, при определении пространства имен redirect Xalan интерпретирует это значение как имя класса Java. Иначе говоря, при обнаружении элемента или функции расширения, определенных в этом пространстве имен, Xalan пытается загрузить класс org.apache.xalan.xslt.extensions.Redirect.

В первой части таблицы стилей функция element-available() проверяет, поддерживает ли процессор XSLT элемент XSLT 2.0 <result-document>. Во второй части мы сообщаем процессору, что следует делать, а также включаем элемент <xsl:fallback> на случай, если процессор не поддерживает XSLT 2.0.

Первая часть таблицы стилей выглядит так:

```
<xsl:choose>
  <xsl:when test="element-available('xsl:result-document')">
    <-- Создать внешнюю ссылку и пункт списка для каждого элемента <chapter> -->
  </xsl:when>
  <xsl:when test="element-available('redirect:write')">
    <-- Создать внешнюю ссылку и пункт списка для каждого элемента <chapter> -->
  </xsl:when>
  <xsl:otherwise>
    <-- Создать внутреннюю ссылку и пункт списка для каждого элемента
                                                          <chapter> -->
```

```
    </xsl:otherwise>
  </xsl:choose>
```

В первых двух ветвях генерируется ссылка в формате `<a href="chaptern.html">`, где *n* – позиция текущего элемента `<chapter>`. Для последней ветви генерируется ссылка в формате `<a href="#chaptern">`. Если процессор способен сгенерировать несколько выходных файлов, мы генерируем ссылку на этот отдельный файл; в противном случае создается ссылка на секцию текущего документа.

Вторая часть таблицы стилей выглядит так:

```
<xsl:result-document method="html"
  include-content-type="no"
  href="{concat('chapter', position(), '.html')}">
  <-- Создать отдельный файл HTML для каждого элемента <chapter> -->
  <xsl:fallback>
    <redirect:write select="concat('chapter', position(), '.html')">
    <-- Создать отдельный файл HTML для каждого элемента <chapter> -->
      <xsl:fallback>
         <-- Создать именованную якорную ссылку, элемент <h1> и абзац >
      </xsl:fallback>
    </redirect:write>
  </xsl:fallback>
</xsl:result-document>
```

Мы обращаемся к процессору XSLT с запросом на использование элемента `<xsl:result-document>`. Если процессор не поддерживает XSLT 2.0, попытка завершается неудачей, и процессор пытается использовать элемент `<xsl:fallback>`. Если используется Xalan-J, элемент `<redirect:write>` поддерживается; если нет, процессор использует последний элемент `<xsl:fallback>`.

Если процессор XSLT способен сгенерировать несколько выходных файлов, основной документ HTML выглядит так, как показано на рис. 9.2.

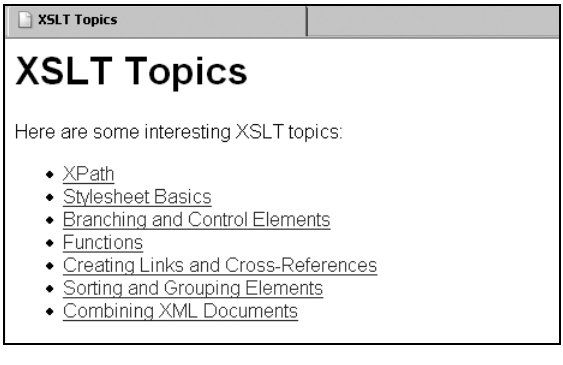


***Рис. 9.2.** Файл HTML со ссылками на внешние файлы*

Щелчок на любой ссылке открывает соответствующий сгенерированный файл отдельной главы (рис. 9.3).

Если процессор XSLT не может сгенерировать несколько выходных файлов, таблица стилей создает один файл, содержащий текст всех элементов <chapter>. Документ начинается с набора ссылок на секции документа (рис. 9.4).

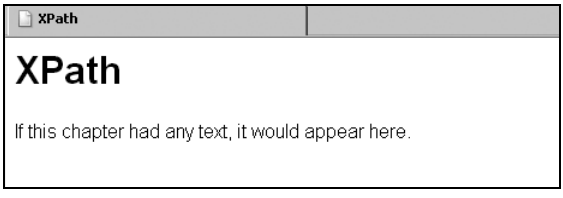


***Рис. 9.3.** Отдельный выходной файл HTML*

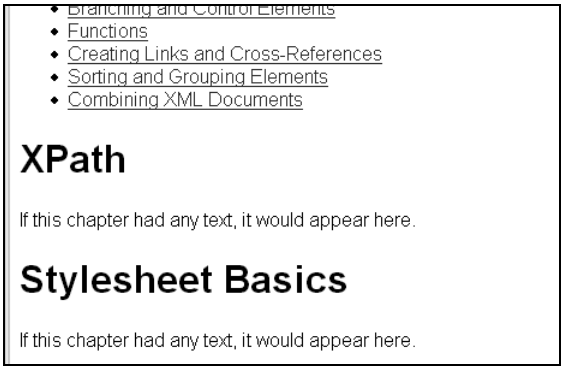


***Рис. 9.4.** Выходной файл HTML с текстом всех глав*

Полный код таблицы стилей:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- multiple-output-files.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:redirect="org.apache.xalan.xslt.extensions.Redirect"
  extension-element-prefixes="redirect">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <xsl:comment>
          <xsl:text>Results generated by </xsl:text>
          <xsl:value-of select="system-property('xsl:vendor')"/>
        </xsl:comment>
```

```
      <title><xsl:value-of select="book/title"/></title>
    </head>
    <body style="font-family: sans-serif;">
      <h1><xsl:value-of select="book/title"/></h1>
      <p>Here are some interesting XSLT topics:</p>
      <ul>
        <xsl:choose>
          <xsl:when test="element-available('xsl:result-document')">
           <xsl:apply-templates select="book/chapter" mode="separate-files"/>
          </xsl:when>
          <xsl:when test="element-available('redirect:write')">
           <xsl:apply-templates select="book/chapter" mode="separate-files"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:for-each select="book/chapter">
              <li>
                <a href="{concat('#chapter', position())}">
                  <xsl:value-of select="title"/>
                </a>
              </li>
            </xsl:for-each>
          </xsl:otherwise>
        </xsl:choose>
      </ul>

      <xsl:for-each select="book/chapter">
        <xsl:result-document method="html"
          include-content-type="no"
          href="{concat('chapter', position(), '.html')}">
          <xsl:apply-templates select="." mode="create-html-file"/>

          <xsl:fallback>
            <redirect:write
              select="concat('chapter', position(), '.html')">
              <xsl:apply-templates select="." mode="create-html-file"/>

              <xsl:fallback>
                <a name="{concat('chapter', position())}"/>
                <h1>
                  <xsl:value-of select="title"/>
                </h1>
                <xsl:apply-templates select="*[position() > 1]"/>
              </xsl:fallback>
            </redirect:write>
          </xsl:fallback>
        </xsl:result-document>
      </xsl:for-each>
    </body>
  </html>
</xsl:template>

<xsl:template match="chapter" mode="separate-files">
  <li>
    <a href="{concat('chapter', position(), '.html')}">
```

```
        <xsl:value-of select="title"/>
      </a>
    </li>
  </xsl:template>

  <xsl:template match="chapter" mode="create-html-file">
    <html>
      <head>
        <title><xsl:value-of select="title"/></title>
      </head>
      <body style="font-family: sans-serif;">
        <h1><xsl:value-of select="title"/></h1>
        <xsl:apply-templates select="*[position() > 1]"/>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Обратите внимание: в таблице стилей указана версия 2.0. Если сообщить процессору XSLT 1.0, что таблица стилей должна обрабатываться по правилам XSLT 2.0, процессор использует режим прямой совместимости. В этом режиме игнорируются некоторые конструкции, ошибочные в XSLT 1.0, а при обнаружении в пространстве имен XSLT неопознанного элемента процессор переходит к элементу `<xsl:fallback>`.

Если изменить таблицу стилей и указать в ней атрибут `version="1.0"`, Xalan (или MSXSL, или любой другой используемый процессор XSLT 1.0) выдаст фатальную ошибку, потому что элемент `<xsl:result-document>` в XSLT 1.0 не определен. Если вы используете *какие-либо* средства XSLT 2.0, проследите за тем, чтобы в элемент `<xsl:stylesheet>` был явно включен атрибут `version="2.0"`.

В этом относительно простом примере один документ XML разбивается на несколько файлов HTML, для всех файлов генерируются осмысленные имена и в основной документ включаются гиперссылки на разные файлы HTML. До сих пор речь шла исключительно об *использовании* существующих расширений; в следующей части главы вы узнаете, как написать собственное расширение.

Создание пользовательских схем упорядочения

В спецификации XSLT 2.0 неоднократно упоминаются *схемы упорядочения* (*collations*), определяющие способ сортировки и сравнения символов. В английском языке не существует символов с диакритическими знаками и символьных последовательностей, сортируемых как отдельные буквы, поэтому для документов на английском языке проблем с сортировкой и сравнением символов нет. Но даже если английский является вашим родным языком, вполне возможно, что вам придется работать с документами на других языках. В этом случае такие симво-

лы, как испанский *ch* (считается отдельной буквой *che*), или символы с диакритическими знаками (например, немецкий «u-умляут», который может записываться в виде *ü* или *ue*) начинают играть важную роль при сортировке и сравнении слов.

По аналогии с функциями расширения спецификация XSLT 2.0 определяет атрибуты, обозначающие возможность применения пользовательских схем упорядочения, но не определяет способ идентификации фрагмента кода, выполняющего эту работу. Поскольку Saxon является лидером в области реализации этих функций, мы сосредоточимся на работе с пользовательскими схемами упорядочения в Saxon. Для начала рассмотрим две схемы упорядочения. Первая сортирует испанские слова так, чтобы комбинация *ch* сортировалась как отдельная буква между *c* и *d*. Вторая схема сравнивает немецкие слова так, что *Müller* и *Mueller* считаются идентичными.

Автор не владеет ни испанским, ни немецким; пожалуйста, простите ему любые неверные утверждения, относящиеся к этим языкам. Этот раздел всего лишь показывает, как создать расширение, реализующее пользовательскую схему упорядочения, а потом использовать его для сортировки и сравнения текста в таблицах стилей.

В традиционной испанской схеме упорядочения, которую мы реализуем, комбинации *ch*, *ll* и *ñ* считаются отдельными буквами, следующими в порядке сортировки после *c*, *l* и *n* соответственно. Однако во многих испано-говорящих странах сейчас используется современная схема упорядочения, определенная Ассоциацией академий испанского языка (La Asociación de Academias de Lingua Española). Современная схема упорядочения не рассматривает *ch* и *ll* как специальные символы; при сортировке они занимают те же места, что и по правилам английского языка. Буква *ñ*, как и прежде, следует за буквой *n*.

В немецком языке при сортировке применяются три схемы упорядочения: DIN-1, DIN-2 и Austrian. (Стандарты DIN определяются комитетом стандартизации Deutsches Institut Für Normung.) Выбор схемы упорядочения зависит от страны. Как правило, DIN-1 используется для сортировки слов, хотя в Швейцарии эта схема также применяется при сортировке имен. DIN-2 – алгоритм упорядочения, реализуемый ниже, – используется для сортировки имен в Германии. В Австрии используется схема упорядочения Austrian, хотя в последнее время от нее постепенно отказываются в пользу правил DIN-2. Основные сложности с алгоритмом DIN-2 связаны с тем, что символ *ä* эквивалентен *ae*, символ *ö* эквивалентен *oe*, символ ß эквивалентен *ss*, а символ *ü* эквивалентен *ue*. Наш код должен «понимать», что два символа в одном слове могут быть эквивалентны одному символу в другом слове.

По моему мнению, приводимые ниже примеры являются правильными реализациями традиционного испанского алгоритма упорядочения и алгоритма DIN-2.

Использование при сортировке пользовательской схемы упорядочения

Начнем с короткого списка слов для сортировки:

```
<?xml version="1.0"?>
<!-- words.xml -->
<wordlist>
  <word>campo</word>
  <word>luna</word>
  <word>ciudad</word>
  <word>llaves</word>
  <word>chihuahua</word>
  <word>arroz</word>
  <word>limonada</word>
</wordlist>
```

Порядок сортировки испанских слов в этом документе отличается от порядка по правилам английского языка. В нашей таблице стилей для демонстрации функций расширения используются два элемента `<xsl:template>`:

```
<?xml version="1.0"?>
<!-- custom-collation1.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Test of sorting with custom collations:&#xA;&#xA;</xsl:text>
    <xsl:variable name="items" as="xs:string*" select="wordlist/word"/>
    <xsl:text>Word list in original order:&#xA;&#xA;</xsl:text>
    <xsl:value-of select="$items" separator="&#xA;"/>
    <xsl:text>&#xA;</xsl:text>
    <xsl:call-template name="ascending-alpha-sort">
      <xsl:with-param name="items" select="$items"/>
    </xsl:call-template>
    <xsl:call-template name="spanish-alpha-sort">
      <xsl:with-param name="items" select="$items"/>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="ascending-alpha-sort">
    <xsl:param name="items"/>
    <xsl:text>&#xA;Ascending text sort:&#xA;</xsl:text>
    <xsl:for-each select="$items">
      <xsl:sort/>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>
```

```
  <xsl:template name="spanish-alpha-sort">
    <xsl:param name="items"/>
    <xsl:text>&#xA;Spanish alpha sort:&#xA;</xsl:text>
    <xsl:for-each select="$items">
      <xsl:sort
        collation="{concat('http://saxon.sf.net/collation?',
                   'class=com.oreilly.xslt.SpanishCollation;')}"/>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

В значении атрибута `collation` приходится включать шаблон значения атрибута, потому что мы используем функцию `concat()` для размещения листинга на странице. Если бы все значение было записано в одной строке (`collation="http://saxon.sf.net....`), фигурные скобки были бы не нужны. Если вас не интересует, как таблица стилей будет выглядеть при выводе, просто поместите весь текст в одной длинной строке – в этом случае `concat` и фигурные скобки становятся лишними.

Для идентификации класса, реализующего пользовательскую схему упорядочения, будет использоваться схема Saxon. (Функция `concat()` объединяет две половины строки, чтобы листинг нормально размещался на странице; обычно 70 символов в значении атрибута записываются в одну строку.) В Saxon значение атрибута `collation` представляет собой URL-адрес, завершаемый вопросительным знаком и ключевым словом `class=`. Все, что следует за `class=`, интерпретируется как имя класса Java. Если класс не удается найти или загрузить, происходит фатальная ошибка. Код расширения, реализующего пользовательскую схему упорядочения:

```
/**
 * SpanishCollation.java
 */

package com.oreilly.xslt;

import java.text.ParseException;
import java.text.RuleBasedCollator;

public class SpanishCollation extends RuleBasedCollator
{
  public SpanishCollation() throws ParseException
  {
    super(traditionalSpanishRules);
  }

  private static String smallnTilde  = new String("\u00F1");
  private static String capitalNTilde = new String("\u00D1");

  private static String traditionalSpanishRules =
```

```
        ("< a,A < b,B < c,C < ch, cH, Ch, CH "  +
         "< d,D < e,E < f,F < g,G < h,H < i,I " +
         "< j,J < k,K < l,L < ll, lL, Ll, LL "  +
         "< m,M < n,N " +
         "< " + smallnTilde + ",' + capitalNTilde + " " +
         "< o,O < p,P < q,Q < r,R < s,S < t,T " +
         "< u,U < v,V < w,W < x,X < y,Y < z,Z");
    }
```

Строка `traditionalSpanishRules` содержит правила сравнений символов. Обратите внимание на определение особых случаев для *ch*, *ll*, $\tilde{N}$ и $\tilde{n}$. Чтобы создать пользовательскую схему упорядочения, достаточно написать строку, определяющую правила упорядочения, и передать ее конструктору класса `RuleBasedCollator`. За дополнительной информацией о классе `java.text.RuleBasedCollator` обращайтесь к документации JDK.

Если применить таблицу стилей к исходному документу, таблица сначала выводит слова в порядке их следования в документе, а затем вызывает два шаблона для сортировки слов. Первый шаблон использует схему упорядочения по умолчанию, так что слова сортируются по правилам английского языка. (Английская схема упорядочения установлена по умолчанию на *моем* компьютере; на вашем компьютере может быть выбрана другая схема.) Второй шаблон вызывает класс Java для сравнения символов. Результаты для этих двух шаблонов различаются:

```
Test of sorting with custom collations:

Word list in original order:

campo
luna
ciudad
llaves
chihuahua
arroz
limonada

Ascending text sort:
arroz
campo
chihuahua
ciudad
limonada
llaves
luna

Spanish alpha sort:
arroz
campo
ciudad
chihuahua
limonada
luna
llaves
```

Как видите, сортировка списка слов с применением класса `SpanishCollation` дает результат, отличный от результата сортировки по правилам английского языка. Слово `chihuahua` оказывается после `ciudad`, а `llaves` – после `luna`.

Пользовательские схемы упорядочения при сравнении текста

Пользовательские схемы упорядочения могут также применяться при сравнении текста. Конечно, сравнение строк является составной частью сортировки, но в XPath 2.0 and XQuery 1.0 появилась функция `compare()` для сравнения двух строк. Возьмем пример из спецификации XSLT 2.0: в немецком языке слово *улица* может записываться в двух вариантах: *Strasse* и *Straße*. При простом символьном сравнении эти две строки различаются; мы воспользуемся функцией упорядочения, которая распознает эти два слова как равные. Полный код:

```
/**
 * GermanCollation.java
 */
package com.oreilly.xslt;

import java.text.ParseException;
import java.text.RuleBasedCollator;

public class GermanCollation extends RuleBasedCollator
{
  public GermanCollation() throws ParseException
  {
    super(traditionalGermanRules);
  }

  private static String sharpS  = new String("\u00DF");
  private static String uppercaseUmlautA = new String("\u00C4");
  private static String lowercaseUmlautA = new String("\u00E4");
  private static String uppercaseUmlautO = new String("\u00D6");
  private static String lowercaseUmlautO = new String("\u00F6");
  private static String uppercaseUmlautU = new String("\u00DC");
  private static String lowercaseUmlautU = new String("\u00FC");

  private static String traditionalGermanRules =
    ("< a,A " +
     "<" + lowercaseUmlautA + "=ae " +
     "<" + uppercaseUmlautA + "=AE " +
     "< b,B < c,C < d,D < e,E < f,F " +
     "< g,G < h,H < i,I < j,J < k,K " +
     "< l,L < m,M < n,N < o,O " +
     "<" + lowercaseUmlautO + "=oe " +
     "<" + uppercaseUmlautO + "=OE " +
     "< p,P < q,Q < r,R < s,S " +
     "< ss=" + sharpS +
     "< t,T < u,U " +
```

```
        "<" + lowercaseUmlautU + "=ue " +
        "<" + uppercaseUmlautU + "=UE " +
        "< v,V < w,W < x,X < y,Y < z,Z");
}
```

В этом коде некоторые строки определяются как равные. Существует семь специальных правил, *определяемых* в нашем классе; каждое правило определяется со знаком равенства в строке `traditionalGermanRules`**. (В классе** `SpanishCollation` **для обозначения порядка сортировки использовался знак** `>`**.) Следующая таблица стилей сравнивает два варианта написания слова «улица» в немецком языке:**

```
<?xml version="1.0"?>
<!-- custom-collation2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="string1" select="'Stra&#xDF;e'"/>
    <xsl:variable name="string2" select="'Strasse'"/>

    <xsl:text>&#xA;Here is a test of the compare() </xsl:text>
    <xsl:text>function, using &#xA;</xsl:text>
    <xsl:text>  an extension function to compare German </xsl:text>
    <xsl:text>characters:&#xA;&#xA;</xsl:text>

    <xsl:text>  compare('</xsl:text>
    <xsl:value-of select="$string1"/>
    <xsl:text>', '</xsl:text>
    <xsl:value-of select="$string2"/>
    <xsl:text>') = </xsl:text>
    <xsl:value-of select="compare($string1, $string2)"/>
    <xsl:text>&#xA;</xsl:text>

    <xsl:text>  compare('</xsl:text>
    <xsl:value-of select="$string1"/>
    <xsl:text>', '</xsl:text>
    <xsl:value-of select="$string2"/>
    <xsl:text>', [German collation]) = </xsl:text>
    <xsl:value-of
      select="compare($string1, $string2,
              concat('http://saxon.sf.net/collation?',
                     'class=com.oreilly.xslt.GermanCollation;'))"/>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here is a test of the compare() function, using
  an extension function to compare German characters:
```

```
compare('Straße', 'Strasse') = 1
compare('Straße', 'Strasse', [German collation]) = 0
```

В функции `compare()` значение `1` указывает, что первая строка находится в порядке сортировки после второй; значение `-1` – что первая строка находится перед второй, а значение `0` – что две строки равны. Из приведенных результатов видно, что при немецкой схеме упорядочения строки *Straße* и *Strasse* равны, несмотря на то, что они явно состоят из разных символов.

Генерирование замаскированной графики

Одну из распространенных разновидностей злоупотреблений в Сети составляют сценарии, которые пытаются создавать десятки учетных записей электронной почты или сотнями закупать билеты на концерты, разбирая формы HTML и обрабатывая их содержимое. Во избежание подобных действий на многих сайтах появились замаскированные графические изображения, содержащие слово с неким визуальным «шумом» (в нашем примере – линиями, проходящими через текст); прочитать такое слово (а следовательно, и работать с формой) может только человек. Мы рассмотрим функцию расширения XSLT, которая по заданному слову генерирует замаскированную графику и веб-страницу.

Для заданного кодового слова наша функция расширения строит графическое изображение и форму HTML, которая предлагает пользователю ввести замаскированное слово. (Конечно, в полноценном решении должен использоваться серверный код, временные cookies и другие средства обработки форм, но эти вопросы выходят за рамки рассматриваемой темы. Сейчас нас интересует лишь расширение XSLT для создания графики.)

Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- hidden-word-test.xsl -->
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:saxon="java:com.oreilly.xslt.HiddenWord"
  xmlns:xalan="xalan://com.oreilly.xslt.HiddenWord"
  xmlns:ora="http://www.oreilly.com/xslt"
  extension-element-prefixes="saxon xalan ora">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>
          Test of hidden word generator
        </title>
      </head>
      <body style="font-family: sans-serif;">
```

```
        <h1>Test of hidden word generator</h1>
      <xsl:comment>
        This <form> doesn't have a method or action
        attribute; a real form would, of course...
      </xsl:comment>
      <xsl:variable name="createGraphic">
        <xsl:choose>
          <xsl:when test="function-available('saxon:createJPEG')">
            <xsl:value-of select="saxon:createJPEG('hidden.jpg',
                                  'giraffe', 48, 200, 100)"/>
          </xsl:when>
         <xsl:when test="function-available('xalan:createJPEG')">
          <xsl:value-of select="xalan:createJPEG('hidden.jpg',
                                'monkey', 48, 200, 100)"/>
        </xsl:when>
        <xsl:when test="function-available('ora:createJPEG')">
          <xsl:value-of select="ora:createJPEG('hidden.jpg',
                               'okapi', 48, 200, 100)"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="1"/>
        </xsl:otherwise>
       </xsl:choose>
      </xsl:variable>
      <form align="center">
        <p>Enter the word hidden in the graphic below:</p>
        <xsl:choose>
          <xsl:when test="number($createGraphic)">
            <p style="font-style: italic; font-size: 150%;">
             <xsl:text>Sorry, the image function isn't available.</xsl:text>
            </p>
          </xsl:when>
          <xsl:otherwise>
            <p>
              <img src="hidden.jpg" width="200" height="100" border="3"/>
            </p>
          </xsl:otherwise>
        </xsl:choose>
        <p>
          <input type="text" name="hiddenWord" size="20" maxlength="20"/>
        </p>
        <p>
          <input type="submit" value="Submit"/>
        </p>
      </form>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Таблица стилей проверяет доступность разных версий расширения функцией `function-available()`**. Если используется процессор Saxon,**

первый элемент <xsl:when> возвращает true; вторая ветвь истинна для процессора Xalan, а третья – для процессора .NET. Конечно, это подразумевает доступность соответствующих библиотек. Если среда времени выполнения Java не может найти класс, function-available() возвращает false.

Обратите внимание: элемент <xsl:choose> находится внутри <xsl:variable>. Наша функция расширения возвращает 0, если функция доступна и работает корректно. Если ни одна из функций не доступна, переменная имеет значение 1; сами функции могут вернуть ненулевой код, если что-то пойдет не так.

Размещение этого кода в переменной позволяет функциям расширения вернуть значение, которое не будет записано в выходной документ. В нижней части таблицы стилей тест number($createGraphic) преобразует значение, возвращаемое функцией расширения, в число. Если значение отлично от нуля, выводится сообщение об ошибке. В противном случае строится форма, содержащая сгенерированную графику.

Версия для Java

Для двух процессоров на базе Java применяется одинаковый код; таблица стилей просто использует правила Saxon и Xalan для идентификации класса расширения. Функция расширения получает во входных данных кодовое слово, имя создаваемого файла JPEG, размер шрифта текста и ширину с высотой изображения в формате JPEG. Получив эти параметры, функция выбирает случайный шрифт и создает пустой холст заданного размера. Затем код Java проверяет, что выбранный шрифт существует (Java работает на многих платформах, не входящих в семейство Windows, поэтому необходима дополнительная проверка.) После назначения шрифта функция вычисляет размеры кодового слова, записанного этим шрифтом заданной величины. Если слово по ширине не умещается в графическое изображение, мы уменьшаем размер шрифта на 2 пункта до тех пор, пока слово не поместится внутри холста.

После назначения шрифта и размера шрифта слово выводится на холсте, а картина завершается рисованием случайных линий. Когда вывод будет закончен, пара строк кода Java записывает содержимое холста в файл JPEG. В нескольких завершающих строках обрабатываются все возникающие исключения. Полный листинг выглядит так:

```
/*
 * HiddenWord.java
 */

package com.oreilly.xslt;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Font;
```

```
import java.awt.FontMetrics;
import java.awt.Graphics2D;
import java.awt.GraphicsEnvironment;
import java.awt.Rectangle;
import java.awt.font.LineMetrics;
import java.awt.image.BufferedImage;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Random;

import com.sun.image.codec.jpeg.ImageFormatException;
import com.sun.image.codec.jpeg.JPEGCodec;
import com.sun.image.codec.jpeg.JPEGImageEncoder;

// Класс создает файл JPEG с замаскированным словом.
public class HiddenWord
{
  // Три шрифта, обычно присутствующих в любой системе...
  // Если замаскированные слова должны генерироваться на другом языке,
  // имена шрифтов необходимо изменить.
  public static String[] fontNames = {"Arial", "Times", "Verdana"};

  // createJPEG() - имя функции расширения.
  public static int createJPEG(String secretWord,
                               String outputFilename,
                               Double dFontSize,
                               Double dWidth, Double dHeight)
    throws IOException, FileNotFoundException,
      ImageFormatException, Exception
  {
    int rc = 0;

    try
    {
      int fontSize = dFontSize.intValue();
      int width = dWidth.intValue();
      int height = dHeight.intValue();

      // Создание нового объекта BufferedImage, который будет использоваться
      // в качестве холста. Кодовое слово с маскирующими линиями выводятся
      // на холсте, после чего результат записывается в файл JPEG.
      BufferedImage bi =
        new BufferedImage(width, height, BufferedImage.TYPE_3BYTE_BGR);

      Random r = new Random();
      String fontName = fontNames[r.nextInt(fontNames.length)];

      // Графическое изображение заполняется белым фоном.
      Graphics2D g = bi.createGraphics();
      g.setColor(Color.WHITE);
      g.fill(new Rectangle(0, 0, width, height));
      g.setColor(Color.BLACK);

      int fontStyle = Font.BOLD;
      int textWidth = 0;
```

```
int textHeight = 0;

// Теперь шрифт необходимо загрузить. Существует вероятность,
// что выбранный шрифт отсутствует в системе, поэтому мы просматриваем
// список шрифтов в поисках нужного. Если искомый шрифт недоступен,
// используется Arial. Такая процедура выбора шрифта намного сложнее,
// чем в версии для .NET, где можно считать, что некоторые шрифты
// заведомо установлены на любом компьютере с системой Windows.
GraphicsEnvironment ge = GraphicsEnvironment.
                         getLocalGraphicsEnvironment();
Font allFonts[] = ge.getAllFonts();
Font chosenFont = new Font("Arial", fontStyle, fontSize);
g.setFont(chosenFont);
FontMetrics fm = g.getFontMetrics();

boolean fontNotFound = true;
int j = 0;
while (fontNotFound && (j < allFonts.length))
{
  if (allFonts[j].getFontName().contains(fontName))
  {
    chosenFont = allFonts[j].deriveFont(fontStyle, fontSize);
    if (!chosenFont.getFontName().equalsIgnoreCase(fontName))
    {
      fontStyle = Font.PLAIN;
      chosenFont = allFonts[j].deriveFont(fontStyle, fontSize);
    }
    g.setFont(chosenFont);
    fm = g.getFontMetrics();

    // Вычисление ширины и высоты слова, выведенного текущим шрифтом.
    // Если результат не помещается на графическом холсте,
    // уменьшаем размер шрифта и пробуем снова.
    textWidth = fm.stringWidth(secretWord);
    while (textWidth > width && fontSize > 2)
    {
      fontSize -= 2;
      chosenFont = allFonts[j].deriveFont(fontStyle, fontSize);
      g.setFont(chosenFont);
      fm = g.getFontMetrics();
      textWidth = fm.stringWidth(secretWord);
    }
    if (fontSize < 1)
      chosenFont = allFonts[j].deriveFont(fontStyle, 12);

    g.setFont(chosenFont);
    fontNotFound = false;
  }
  else
    j++;
}

// Вывод строки на холсте. Для выравнивания текста по центру
// холста используются размеры самого холста и размеры строки.
```

```
        fm = g.getFontMetrics(chosenFont);
        LineMetrics lm =
          chosenFont.getLineMetrics(secretWord, g.getFontRenderContext());
        textHeight = (int)lm.getAscent();
        textWidth = fm.stringWidth(secretWord);
        g.drawString(secretWord, (width - textWidth) / 2, textHeight + 30);

        // Рисуем случайные линии, чтобы замаскировать текст.
        g.setStroke(new BasicStroke((float)2.0));

        for (int i = 0; i < width / 30; i++)
          g.drawLine(0, r.nextInt(height), width, r.nextInt(height));

        int numLines = java.lang.Math.max(height, width) / 30;
        for (int i = 0; i < numLines; i++)
        {
          int nextX = r.nextInt(width);
          g.drawLine(nextX, 0, nextX, height);
          int nextY = r.nextInt(height);
          g.drawLine(0, nextY, width, nextY);
        }

        for (int i = 0; i < height / 20; i++)
          g.drawLine(r.nextInt(width), 0, r.nextInt(width), height);

        // Мы нарисовали на холсте все, что требовалось.
        // Теперь содержимое холста записывается в файл JPEG.
        FileOutputStream fos = new FileOutputStream(outputFilename);
        JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(fos);
        encoder.encode(bi);
        fos.flush();
        fos.close();
      }
      catch (FileNotFoundException fnfe)
      {
        rc = 4;
        System.err.println(fnfe);
      }
      catch (IOException ioe)
      {
        rc = 8;
        System.err.println(ioe);
      }
      catch (Exception e)
      {
        rc = 12;
        System.err.println(e);
      }
      return rc;
    }
}
```

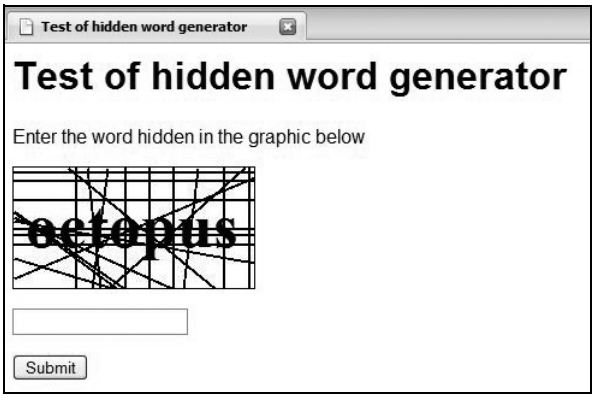


Рис. 9.5. *Форма HTML с замаскированным словом*

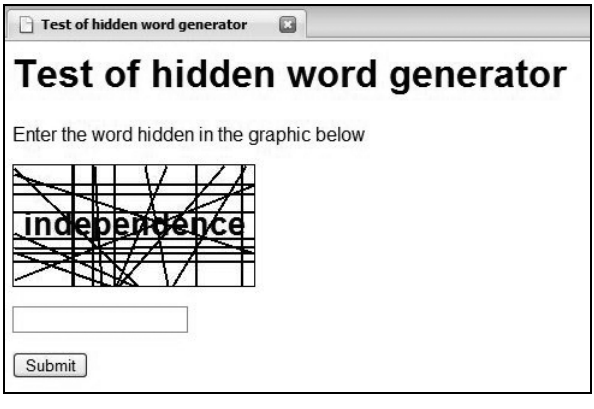


Рис. 9.6. *Форма HTML с замаскированным словом большей длины*

Сгенерированная форма HTML показана на рис. 9.5.

На рис. 9.6 изображена замаскированная графика с кодовым словом большей длины; рисунок показывает, как работает механизм изменения размеров текста. Как видите, шрифт уменьшился, чтобы слово поместилось в области холста.

Версия для .NET

Версия для .NET работает аналогично, хотя для того, чтобы использовать расширение, нам понадобятся два файла C#. Первый файл создает объект `XslCompiledTransform` и связывает с ним функцию расширения. Второй файл содержит фактическую реализацию расширения и создает замаскированное изображение.

Первый файл C#, создающий объект преобразования:

```
/*
 * Main.cs
 */

using System;
using System.Text;

using System.Xml;
using System.Xml.Xsl;

namespace com.oreilly.xslt
{
  class MainClass
  {
    static void Main(string[] args)
    {
      if (args.Length < 3)
      {
        System.Console.WriteLine("\nUsage: HiddenWordExample xml-file " +
          "xsl-file output-file");
        System.Console.WriteLine("\n  Example: HiddenWordExample " +
          "blank.xml hidden-word-test.xsl results.html");
      }
      else
      {
        // Создание объекта таблицы стилей и объекта XMLWriter,
        // который записывает выходные данные в файл.
        XslCompiledTransform stylesheet = new XslCompiledTransform();
        XmlTextWriter xWriter =
            new XmlTextWriter(args[2], Encoding.UTF8);

        // Использование объекта XsltSettings, который разрешает
        // выполнение сценариев (что необходимо для расширений),
        // а затем загружает таблицу стилей.
        XsltSettings settings = new XsltSettings(true, true);
        stylesheet.Load(args[1], settings, new XmlUrlResolver());

        // Создание объекта расширения.
        HiddenWord hw = new HiddenWord();

        // Создание функции расширения с использованием
        // объекта XsltArgumentList.
        XsltArgumentList argList = new XsltArgumentList();
        argList.AddExtensionObject("http://www.oreilly.com/xslt", hw);

        // Когда подготовка будет завершена, вызываем метод Transform(),
        // который и выполняет всю работу...
        stylesheet.Transform(args[0], argList, xWriter);
      }
    }
  }
}
```

Файл C#, создающий объект `XslCompiledTransform`**, обязан определить объект реализации функции расширения (экземпляр класса** `Hidden-`

Word) и связать его с пространством имен (http://www.oreilly.com/xslt). Пространство имен, определенное в коде C#, должно соответствовать пространству имен, определяемому в таблице стилей. При запуске этого кода, оформленного в виде файла *.exe*, должны передаваться аргументы с именами трех файлов: исходного файла XML, таблицы стилей и файла результата. Если пользователь не передаст как минимум три аргумента, программа выведет сообщение об ошибке, предотвращая исключение IndexOutOfRange.

Итак, мы разобрались с подготовкой преобразования; далее приводится код объекта расширения. Хотя имена методов и системных объектов отличаются от версии для Java, общая структура кода остается неизменной:

```
/*
 * HiddenWord.cs
 */

using System;
using System.Text;
using System.Xml;
using System.Xml.Xsl;
using System.Drawing;

// Класс создает файл JPEG с замаскированным словом.

namespace com.oreilly.xslt
{
  class HiddenWord
  {
    // Три шрифта, обычно присутствующих в системе Windows...
    // Если замаскированные слова должны генерироваться на другом языке,
    // имена шрифтов необходимо изменить.
    private static String[] fontNames = { "Arial", "Times", "Verdana" };

    // createJPEG() - имя функции расширения.
    public int createJPEG(String outputFilename,
                          String secretWord, Double dFontSize,
                          Double dWidth, Double dHeight)
    {
      int rc = 0;

      int fontSize = (int)dFontSize;
      int width = (int)dWidth;
      int height = (int)dHeight;

      Random r = new Random();

      String fontName = fontNames[r.Next(fontNames.Length)];

      // Создание нового объекта Bitmap. Текст и графика
      // выводятся на объекте Bitmap, после чего
      // результат записывается в файл JPEG.
      Bitmap pic = new Bitmap(width, height);
      Graphics context = Graphics.FromImage(pic);
```

```
    // Графическое изображение заполняется белым фоном.
    SolidBrush brush = new SolidBrush(Color.White);
    context.FillRectangle(brush, 0, 0, width, height);

    // Загрузка случайно выбранного шрифта.
    Font currentFont = new Font(fontName, fontSize, FontStyle.Bold);

    // Вычисление ширины и высоты слова, выведенного текущим шрифтом.
    // Если результат не помещается в границах Bitmap,
    // уменьшаем размер шрифта и пробуем снова.
    SizeF textSize = context.MeasureString(secretWord, currentFont);
    float textWidth = textSize.Width;

    while (textWidth > width && fontSize > 2)
    {
      fontSize -= 2;
      currentFont = new Font(fontName, fontSize, FontStyle.Bold);
      textSize = context.MeasureString(secretWord, currentFont);
      textWidth = textSize.Width;
    }
    if (fontSize < 1)
      currentFont = new Font(fontName, 12, FontStyle.Bold);

    float textHeight = textSize.Height;
    brush.Color = Color.Black;

    // Вывод строки на холсте. Для выравнивания текста
    // по центру холста используются размеры самого холста
    // и размеры строки, выводимой текущим шрифтом.
    context.DrawString(secretWord, currentFont, brush,
        (width - textWidth) / 2, (height - textHeight) / 2);

    // Рисуем линии, чтобы замаскировать текст. Координаты
    // начала и конца линии определяются случайным образом.
    Pen pen = new Pen(Color.Black, (float)3);
    for (int i = 0; i < width / 30; i++)
      context.DrawLine(pen, 0, r.Next(height), width, r.Next(width));

    int numLines = Math.Max(height, width) / 30;
    for (int i = 0; i < numLines; i++)
    {
      int nextX = r.Next(width);
      context.DrawLine(pen, nextX, 0, nextX, height);
      int nextY = r.Next(height);
      context.DrawLine(pen, 0, nextY, width, nextY);
    }

    for (int i = 0; i < height / 20; i++)
      context.DrawLine(pen, r.Next(width), 0, r.Next(width), height);

    // Мы нарисовали на объекте Bitmap все, что требовалось.
    // Теперь его содержимое записывается в файл JPEG.
    pic.Save(outputFilename, System.Drawing.Imaging.ImageFormat.Jpeg);

    return rc;
  }
```

```
    }
  }
```

Как видите, код C# строится по тому же образцу, что и код Java. Мы создаем новую область графического вывода, уменьшаем размер шрифта, пока слово не поместится в ней, рисуем текст и маскирующие линии и записываем результат в файл JPEG.

Этот пример показывает, как создать графику в формате JPEG из текста в таблице стилей. Кодовое слово также можно передать в глобальном параметре или воспользоваться данными из исходного документа XML. Как упоминалось ранее, полноценное решение должно генерировать серверный код для проверки слова, введенного пользователем. Также следует учесть, что использованные шрифты не подойдут для кодовых слов, записанных на других языках (русском, японском, греческом, китайском, корейском и т. д.). В таких случаях можно либо изменить имена шрифтов в программном коде, либо изменить таблицу стилей и функцию расширения так, чтобы имя шрифта передавалось в аргументе из таблицы стилей.

Пример: построение круговых диаграмм в формате SVG

Возможно, вы заметили, что набор математических функций в XPath и XSLT весьма ограничен, даже в XSLT 2.0. В этом примере мы напишем расширение, для работы которого необходимы различные тригонометрические функции. Задача может быть решена несколькими способами; в частности, мы можем использовать:

- Механизм расширения Xalan и Saxon для вызова статических методов класса `java.lang.Math`.
- Тригонометрические функции из библиотеки EXSLT (подробности см. ниже).
- Bean Scripting Framework; эта интересная инфраструктура из проекта Apache Jakarta позволяет писать функции расширения на различных сценарных языках, включая JRuby, JavaScript, Jython и Jacl.
- Классы библиотеки .NET для расширения процессора Microsoft XSLT.

Мы начнем с построения первого варианта таблицы стилей, а затем рассмотрим последовательные изменения во всех последующих итерациях. Наша задача – сгенерировать круговую диаграмму в формате SVG (Scalable Vector) на базе документа XML. Документ содержит информацию об объемах продаж для разных магазинов компании; для построения круговой диаграммы по данным документа необходимо вычислить размеры секторов. Исходный документ XML выглядит так:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate-sales.xml -->
<report month="8" year="2006">
  <caption>
    <heading>Chocolate bar sales</heading>
    <subheading>(units)</subheading>
  </caption>
  <store>
    <name>Carrboro</name>
    <brand name="Lindt">27408</brand>
    <brand name="Callebaut">8203</brand>
    <brand name="Valrhona">22101</brand>
    <brand name="Perugina">14336</brand>
    <brand name="Ghirardelli">19268</brand>
  </store>
  <store>
    <name>Chapel Hill</name>
    <brand name="Lindt">28503</brand>
    <brand name="Valrhona">7287</brand>
    <brand name="Perugina">12077</brand>
    <brand name="Ghirardelli">8392</brand>
    <brand name="Callebaut">17294</brand>
  </store>

...

  </store>
</report>
```

Диаграмма SVG для этого документа показана на рис. 9.7.

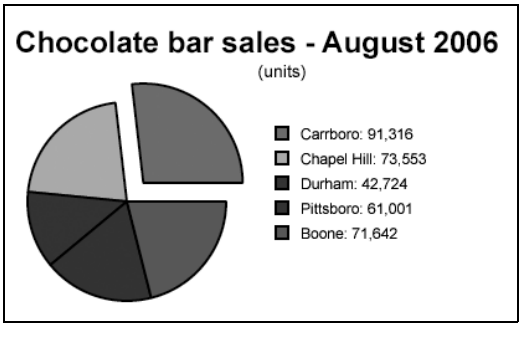


Рис. 9.7. *Круговая диаграмма в формате SVG*

Чтобы диаграмма была действительно полезной, мы воспользуемся сценарными возможностями Adobe SVG Viewer. Мы сгенерируем графику диаграммы, а также функции EcmaScript для отображения дополнительной информации. Например, при наведении указателя мыши на сектор будет выводиться информация по соответствующему магазину (рис. 9.8).

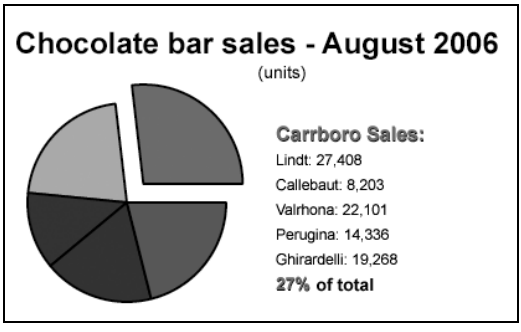


Рис. 9.8. *Круговая диаграмма SVG реагирует на перемещения указателя мыш*

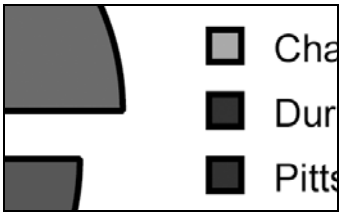


Рис. 9.9. *Круговая диаграмма масштабируется до произвольных размеров*

Функции соединения линий SVG и другие детали придадут нашей диаграмме профессиональный вид. Например, при увеличении диаграммы фигуры, линии и текст сохранят прежнюю четкость (рис. 9.9).

Генерируемый код EcmaScript совместим только с Adobe SVG Viewer; броузер Firefox отображает круговую диаграмму, но не поддерживает интерактивность, обеспечиваемую кодом EcmaScript.

Ограниченный набор математических функций XPath не позволит вычислить размеры дуг, из которых строится диаграмма, поэтому для решения проблемы мы воспользуемся функциями расширения. К счастью, Java предоставляет все необходимые тригонометрические функции в классе `java.lang.Math`. Что еще лучше, Xalan и Saxon позволяют легко загрузить этот класс и выполнить нужные статические методы (`sin()`, `cos()` и `toRadians()`). Для платформы .NET добавлена секция CDATA с кодом EcmaScript, который может интерпретироваться процессором XSLT для .NET.

EcmaScript – название версии JavaScript, стандартизированной Европейской ассоциацией производителей компьютеров (ECMA, European Computer Manufacturers Association). Мы используем термин «EcmaScript», потому что в примерах из спецификации SVG используется тег `<script type="text/ecmascript">` (хотя термины «EcmaScript» и «JavaScript» часто используются как синонимы).

Остальные важные подробности будут упоминаться по мере их появления в таблице стилей. Сначала необходимо объявить префиксы пространств имен, как это было сделано в элементе расширения:

```
<?xml version="1.0"?>
<!-- piechart.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  extension-element-prefixes="xalan-java saxon-java msxsl ora months"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:xalan-java="http://xml.apache.org/xslt/java"
  xmlns:saxon-java="java:java.lang.Math"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt"
  xmlns:ora="http://www.oreilly.com/xslt"
  xmlns:months="http://www.oreilly.com/xslt/months">
```

Префикс пространства имен `xalan-java` связывается со строкой `http://xml.apache.org/xslt/java`; процессору Xalan эта строка нужна для загрузки классов Java и использования их статических методов. Для поддержки Saxon префикс `saxon-java` связывается со строкой `java:java.lang.Math`. Обратите внимание: два процессора применяют разные методы для загрузки классов Java. В спецификации XSLT не определяются правила идентификации кода, загружаемого процессором, по URI-адресам пространств имен, поэтому, скорее всего, у разных процессоров эти правила будут разные.

Мы также определяем пространства имен `msxsl` и `ora`. Пространство имен `msxsl` позволяет включить EcmaScript прямо в таблицу стилей; для обращения к этому коду используется пространство имен `ora`. В завершение определяется пространство имен `months`. Таблица стилей содержит набор элементов `<months:month>`, каждый из которых определяет английское название некоторого месяца.

Далее идут определения метода вывода и пары глобальных переменных:

```
<xsl:output method="xml"/>

<xsl:variable name="totalSales" select="sum(//brand)"/>
<xsl:variable name="stores" select="count(/report/store)"/>
```

Так как формат SVG входит в номенклатуру XML, в таблице стилей выбирается метод вывода `xml`. Переменная `$totalSales` содержит суммарный объем продаж по всем брендам и магазинам, а в переменной `$stores` хранится количество магазинов. Эти значения можно было бы пересчитывать заново при каждом использовании, но проще сохранить их в глобальных переменных.

Далее следует шаблон `match="/"`. Он определяет элемент `<svg>`, содержащий весь документ SVG. Начало шаблона выглядит так:

```
<svg:svg width="450" height="300" version="1.1"
  baseProfile="full"
  xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink"
```

```
xmlns:ev="http://www.w3.org/2001/xml-events">
```

Мы определяем размеры графического изображения и версию SVG, а также некоторые пространства имен, необходимые для работы интерпретатора SVG. Настало время генерировать функции EcmaScript, обеспечивающие интерактивность диаграммы. В исходном состоянии на диаграмме отображаются сектора и условные обозначения. Каждый сектор окрашивается в свой цвет, а в условных обозначениях указываются магазин, связанный с каждым сектором, и суммарный объем продаж по каждому магазину. Кроме того, сектор первого магазина из исходного документа XML визуально отделяется от остальных секторов.

В коде EcmaScript используются две переменные состояния. Первая – логическая переменная `detailsViewOn` – указывает, какая информация выводится в области справа: условные обозначения или подробная информация по отдельному магазину. Вторая – целочисленная переменная `currentStore` – представляет номер магазина, для которого в данный момент выводятся подробные сведения. Реакция диаграммы на события мыши определяется значениями этих двух переменных.

Функции EcmaScript работают следующим образом:

- Когда пользователь наводит указатель мыши на сектор диаграммы:
  - Если в области условных обозначений *не выводится* постоянная информация по конкретному магазину (`detailsViewOn = false`), условные обозначения заменяются подробными данными по магазину, представленному этим сектором.
  - Если в области условных обозначений *выводится* постоянная информация по конкретному магазину (`detailsViewOn = true`), ничего не происходит.
- Когда пользователь выводит указатель мыши за пределы сектора диаграммы:
  - Если в области условных обозначений *не выводится* постоянная информация по конкретному магазину (`detailsViewOn = false`), подробные данные заменяются стандартными условными обозначениями.
  - Если в области условных обозначений *выводится* постоянная информация по конкретному магазину (`detailsViewff = true`), ничего не происходит.
- Когда пользователь щелкает кнопкой мыши на секторе диаграммы:
  - Если в области условных обозначений *не выводится* постоянная информация по конкретному магазину (`detailsViewOn = false`), условные обозначения заменяются подробными данными по магазину, переменной `detailsViewOn` присваивается значение `true`, а переменной `currentStore` присваивается номер текущего магазина.
  - Если в области условных обозначений *выводится* постоянная информация по конкретному магазину (`detailsViewOn = true`),

а номер магазина сектора, на котором был сделан щелчок, *совпадает* с номером текущего магазина, переменной `detailsViewOn` присваивается значение `false`, а переменной `currentStore` присваивается 0.

- Если в области условных обозначений *выводится* постоянная информация по конкретному магазину (`detailsViewOn = true`), но номер магазина сектора, на котором был сделан щелчок, *не совпадает* с номером текущего магазина, переменной `currentStore` присваивается новое значение, а область условных обозначений заменяется подробной информацией по новому магазину.

Короче говоря, пользователь может навести указатель мыши на сектор диаграммы, чтобы просмотреть данные о продажах по соответствующему магазину. Он может также щелкнуть на секторе, чтобы эти данные постоянно выводились в области условных обозначений. В режиме постоянного вывода пользователь может щелкнуть на другом секторе, чтобы заменить выводимые данные новыми, или же снова щелкнуть на текущем секторе, чтобы вернуться к выводу условных обозначений. (Надеюсь, объяснение получилось ясным; чтобы понять, как работает этот код, достаточно немного поэкспериментировать с ним.)

Мы определяем три функции (`mouse_over()`, `mouse_out()` и `click()`) для обработки интересующих нас событий мыши:

```
  <svg:script type="text/ecmascript">
    <xsl:text disable-output-escaping="yes">

detailsViewOn = false;
currentStore = 0;

function mouse_over(selectedItem, totalItems)
{
  if (!detailsViewOn || selectedItem == currentStore)
  {
    for (i = 1; totalItems >= i; i++)
    {
      obj = svgDocument.getElementById("details" + i);
      obj.setAttributeNS(null, "visibility", "hidden");
      obj = svgDocument.getElementById("legend" + i);
      obj.setAttributeNS(null, "visibility", "hidden");
    }

    obj = svgDocument.getElementById("details" + selectedItem);
    obj.setAttributeNS(null, "visibility", "visible");
  }
}

function mouse_out(totalItems)
{
  if (!detailsViewOn)
  {
    for (i = 1; totalItems >= i; i++)
    {
```

```
      obj = svgDocument.getElementById("details" + i);
      obj.setAttributeNS(null, "visibility", "hidden");
      obj = svgDocument.getElementById("legend" + i);
      obj.setAttributeNS(null, "visibility", "visible");
    }
  }
}

function click(selectedItem, totalItems)
{
  if (selectedItem != currentStore)
  {
    currentStore = selectedItem;
    mouse_over(selectedItem, totalItems);
    detailsViewOn = true;
  }
  else
  {
    obj = svgDocument.getElementById("details" + selectedItem);
    if (obj.getAttributeNS(null, "visibility") == "visible")
    {
      detailsViewOn = false;
      currentStore = 0;
    }
    else
    {
      currentStore = selectedItem;
      mouse_over(selectedItem, totalItems);
      detailsViewOn = true;
    }
  }
}
      //      </xsl:text></svg:script>
    <xsl:text>&#xA;</xsl:text>
```

Обратите внимание на атрибут `disable-output-escaping`: он предотвращает возможные сбои парсера XML из-за знаков > и <.

Если вы записываете сценарный код в файле HTML, включите его в комментарий, чтобы броузеры, не поддерживающие сценариев, игнорировали этот код. Конец сценарного кода должен выглядеть примерно так:

```
...
      else
      {
        currentItem = selectedItem;
        mouse_over(selectedItem, totalItems);
        detailsViewOn = true;
      }
    }
  }
  //      -->
```

Двойная косая черта в конце сценария – маркер комментария, указывающий процессору проигнорировать комбинацию --> в конце элемента <xsl:comment>. Без нее некоторые процессоры пытаются обработать завершение комментария и выдают сообщение об ошибке; в конце концов, символы -- очень похожи на оператор декремента. Помните об этом при генерировании сценарного кода в своих таблицах стилей; в противном случае вам придется возиться с поиском трудноуловимых ошибок в некоторых броузерах.

И последнее, что необходимо сказать о построении кода EcmaScript: в некоторых случаях бывает удобнее выделить код в отдельный файл. SVG позволяет разместить код EcmaScript в отдельном файле и ссылаться на него из SVG. (Этот способ использовался в первом издании книги.) Однако с учетом текущего состояния Adobe SVG Viewer гораздо проще разместить код Ecma Script в том же файле.

После всего необходимого кода EcmaScript мы создаем заголовок и подзаголовок круговой диаграммы:

```
<!-- Построение заголовка и подзаголовка.                  -->
<!-- Мы выводим заголовок и подзаголовок на диаграмме     -->
<!-- и создаем элемент <svg:title> в самом документе SVG. -->
<xsl:variable name="titleText">
  <xsl:value-of select="/report/caption/heading"/>
  <xsl:text> - </xsl:text>
  <xsl:choose>
    <xsl:when test="function-available('math:cos')">
      <xsl:variable name="dateString">
        <xsl:text>--</xsl:text>
        <xsl:if test="string-length(/report/@month) = 1">
          <xsl:text>0</xsl:text>
        </xsl:if>
        <xsl:value-of select="/report/@month"/>
        <xsl:text>--</xsl:text>
      </xsl:variable>
      <xsl:value-of
        select="dates-and-times:month-name($dateString)"/>
      <xsl:text> </xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="/report/@month"/>
      <xsl:text>/</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:value-of select="/report/@year"/>
</xsl:variable>

<svg:title>
  <xsl:value-of select="$titleText"/>
</svg:title>
<svg:text font-size="24px" text-anchor="start"
  font-weight="bold" x="10" y="40">
  <xsl:value-of select="$titleText"/>
```

```
</svg:text>
<svg:text font-size="14px" text-anchor="middle" y="60" x="225">
  <xsl:value-of select="/report/caption/subheading"/>
</svg:text>
```

Мы выводим текст на холсте SVG, а также создаем элемент `<svg:title>` в документе SVG. При отображении документа SVG в броузере эта строка станет заголовком документа; она является аналогом элемента HTML `<title>`.

Наконец, все готово к обработке элементов `<store>`. Каждый элемент `<store>` содержит информацию о продажах конкретного магазина. Обработка каждого элемента производится в три этапа:

1. Сначала создается сектор круговой диаграммы. Подробное описание приводится позже; пока достаточно сказать, что на этом этапе вычисляется размер сектора и добавляются соответствующие атрибуты `onmouseover`, `onmouseout` и `onclick`. Это происходит в шаблоне `<xsl:template match="store" mode="pie">`.
2. Затем создается запись данного магазина в области условных обозначений. Она состоит из квадрата, заполненного цветом данного магазина, названия магазина и суммарного объема продаж. Запись выводится на 20 пикселов ниже записи предыдущего магазина. Это происходит в шаблоне `<xsl:template match="store" mode= "legend">`.
3. Наконец, мы создаем подробную информацию о продажах для данного магазина. Это представление включает название магазина, объемы продаж по каждому бренду и текущий процент продаж магазина от общего объема. Все данные в этом представлении выводятся с атрибутом `visiblility="hidden"`. Подробная информация по конкретному магазину остается скрытой до тех пор, пока не будет активизирована событием мыши. Это происходит в шаблоне `<xsl:template match="store" mode="details">`.

Рассмотрим все эти действия более подробно:

1. Сначала необходимо нарисовать круговую диаграмму. Это делается при помощи элемента `<svg:path>`. Мы используем тригонометрические функции для вычисления внешнего угла сектора дуг, а затем средствами SVG рисуем плавную дугу между конечными точками на границе. SVG обеспечивает и такие технические подробности, как соединение линий и заполнение сектора конкретным цветом.
2. Для каждого сектора вычисляются некоторые значения, которые передаются в параметрах шаблона `mode="pie"`. Сначала мы определяем цвет сектора и объем продаж по конкретному магазину. Цвет вычисляется функцией `position()` и оператором `mod`, а объем продаж – функцией `sum()`.
3. Первый сектор визуально выделяется, т. е. «выдвигается» из круговой диаграммы. Для этого он выводится с некоторым смещением. Значение переменной `$explode` задается следующим образом:

```
<xsl:variable name="explode" select="position()=1"/>
```

4. Остается вычислить суммарные объемы продаж по всем предыдущим магазинам. При выводе каждого сектора оси координат сдвигаются на некоторое количество градусов. Величина поворота зависит от того, какая часть общих продаж была прорисована до настоящего момента. Иначе говоря, если мы прорисовали ровно 50% от общего объема, ось будет повернута на 180 градусов (50% от 360). Поворот оси упрощает тригонометрические вычисления. Для суммирования продаж используется ось `preceding-sibling`:

```
<xsl:with-param name="runningTotal"
  select="sum(preceding-sibling::store/brand)"/>
```

5. После вычисления всех нужных переменных вызывается шаблон:

```
<xsl:apply-templates select="." mode="pie">
  <xsl:with-param name="color" select="$color"/>
  <xsl:with-param name="storeSales" select="$storeSales"/>
  <xsl:with-param name="totalSales" select="$totalSales"/>
  <xsl:with-param name="runningTotal"
    select="sum(preceding-sibling::store/brand)"/>
  <xsl:with-param name="stores" select="$stores"/>
  <xsl:with-param name="explode" select="$explode"/>
</xsl:apply-templates>
```

6. Шаблон начинается с вычисления угла текущего сектора в радианах. Здесь впервые используется одна из функций расширения:

```
<xsl:variable name="currentAngle">
  <xsl:choose>
    <xsl:when
      test="function-available('xalan-java:java.lang.Math.toRadians')">
      <xsl:value-of
        select="xalan-java:java.lang.Math.toRadians(
                ($storeSales div $totalSales) * 360.0)"/>
    </xsl:when>
    <xsl:when
      test="function-available('saxon-java:toRadians')">
      <xsl:value-of
        select="saxon-java:toRadians(
                ($storeSales div $totalSales) * 360.0)"/>
    </xsl:when>
    <xsl:when test="function-available('ora:cos')">
      <xsl:value-of
        select="($storeSales div $totalSales) *
                6.28318530717958647692528676655 9"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:message terminate="yes">
        <xsl:text>Sorry, this stylesheet can't generate any </xsl:text>
        <xsl:text>useful results &#xA;  without trigonometric </xsl:text>
        <xsl:text>functions available. </xsl:text>
```

```
      </xsl:message>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable>
```

При вычислении значения переменной используется функция XSLT `function-available()`. Если таблица стилей обрабатывается процессором Xalan, доступны функции из пространства имен `xalan-java`. При использовании Saxon доступны функции из пространства имен `saxon-java`. С процессором .NET доступны функции из пространства имен `ora`. Если все эти функции недоступны, работа процессора завершается – без тригонометрических функций построить круговую диаграмму не удастся.

В переменной `$currentAngle` хранится угол текущего сектора в радианах. Иначе говоря, если текущий магазин обеспечивает 25% объема продаж компании, мы преобразуем 90 градусов в радианы и используем это значение для прорисовки сектора. (Преобразование в радианы необходимо, потому что тригонометрические функции Java работают именно с этими единицами.) Полученное значение используется позднее при вызове `cos()` и `sin()`.

7. Наконец, все готово к прорисовке сектора. Для этого мы используем элемент SVG `<path>`. Выглядит он так (значения его атрибутов рассматриваются ниже):

```
<svg:path
  fill="blue"
  stroke="black"
  stroke-width="2px"
  fillrule="evenodd"
  stroke-linejoin="bevel"
  onmouseout="mouse_out(5);"
  onmouseover="mouse_over(3, 5);"
  onclick="click(3, 5);"
  transform="translate(100,160)  rotate(-219.65188868902763)"
  d="M 80 0 A 80 80 0 0 0 34.3848056509693 -72.23354581041326 L 0 0 Z "/>
```

Этот устрашающий элемент сгенерирован не менее устрашающим фрагментом таблицы стилей:

```
<svg:path fill="{$color}" stroke="black" stroke-width="2px"
  fillrule="evenodd" stroke-linejoin="bevel">
  <xsl:attribute name="onmouseout">
    <xsl:text>mouse_out(</xsl:text>
    <xsl:value-of select="$stores"/>
    <xsl:text>);</xsl:text>
  </xsl:attribute>
  <xsl:attribute name="transform">
    <xsl:choose>
      <xsl:when test="$explode">
        <xsl:text>translate(</xsl:text>
```

```
        <xsl:choose>
          <xsl:when
            test="function-available('xalan-java:java.lang.Math.cos')">
            <xsl:value-of
              select="(xalan-java:java.lang.Math.cos(
                      $currentAngle div 2) * 20) + 100"/>
            <xsl:text>,</xsl:text>
            <xsl:value-of
             select="(xalan-java:java.lang.Math.sin($currentAngle div 2)
                      * -20) + 160"/>
          </xsl:when>
          <xsl:when
            test="function-available('saxon-java:cos')">
            <xsl:value-of
              select="(saxon-java:cos($currentAngle div 2) * 20) + 100"/>
            <xsl:text>,</xsl:text>
            <xsl:value-of
             select="(saxon-java:sin($currentAngle div 2) * -20) + 160"/>
          </xsl:when>
          <xsl:when test="function-available('ora:cos')">
            <xsl:value-of
              select="(ora:cos($currentAngle div 2) * 20) + 100"/>
            <xsl:text>,</xsl:text>
            <xsl:value-of
              select="(ora:sin($currentAngle div 2) * -20) + 160"/>
          </xsl:when>
        </xsl:choose>
        <xsl:text>) </xsl:text>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>translate(100,160) </xsl:text>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:text> rotate(</xsl:text>
    <xsl:value-of select="-1 * (($runningTotal div $totalSales) * 360.0)"/>
    <xsl:text>)</xsl:text>
  </xsl:attribute>
  <xsl:attribute name="onmouseover">
    <xsl:text>mouse_over(</xsl:text>
    <xsl:value-of select="$position"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="$stores"/>
    <xsl:text>);</xsl:text>
  </xsl:attribute>
  <xsl:attribute name="onclick">
    <xsl:text>click(</xsl:text>
    <xsl:value-of select="$position"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="$stores"/>
    <xsl:text>);</xsl:text>
  </xsl:attribute>
```

```
<xsl:attribute name="d">
  <xsl:text>M 80 0 A 80 80 0 </xsl:text>
  <xsl:choose>
    <xsl:when test="$currentAngle > 3.14">
      <xsl:text>1 </xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>0 </xsl:text>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:text>0 </xsl:text>
  <xsl:choose>
   <xsl:when test="function-available('xalan-java:java.lang.Math.cos')">
     <xsl:value-of
       select="xalan-java:java.lang.Math.cos($currentAngle) * 80"/>
     <xsl:text> </xsl:text>
     <xsl:value-of
       select="xalan-java:java.lang.Math.sin($currentAngle) * -80"/>
    </xsl:when>
    <xsl:when test="function-available('saxon-java:cos')">
      <xsl:value-of select="saxon-java:cos($currentAngle) * 80"/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="saxon-java:sin($currentAngle) * -80"/>
    </xsl:when>
    <xsl:when test="function-available('ora:cos')">
      <xsl:value-of
        select="ora:cos($currentAngle) * 80"/>
      <xsl:text> </xsl:text>
      <xsl:value-of
        select="ora:sin($currentAngle) * -80"/>
    </xsl:when>
  </xsl:choose>
  <xsl:text> L 0 0 Z </xsl:text>
</xsl:attribute>
</svg:path>
```

Мы последовательно рассмотрим атрибуты элемента SVG. Атрибут `fill` определяет цвет, который должен использоваться для заполнения фигуры. Атрибут `stroke` определяет цвет линии для рисования контура, а атрибут `stroke-width` определяет толщину этой линии. Атрибуты `fillrule` и `stroke-linejoin` определяют способ заполнения фигуры и соединения линий.

Затем генерируются вызовы сценарных функций. Для третьего сектора (из пяти) вызовы сценарных функций выглядят так:

```
onmouseout="mouse_out(5);"
onmouseover="mouse_over(3, 5);"
onclick="click(3, 5);"
```

Функция `mouse_out` получает общее количество секторов в круговой диаграмме. Все элементы создаются скрытыми, вследствие чего все

элементы условных обозначений (т. е. элементы с идентификаторами `legend1`, `details1` и т. д.) видны. Функция `mouse_over` получает номер текущего сектора (того, над которым находится указатель мыши) и количество секторов. Функция `click` также получает значение текущего сектора и количество секторов.

Настоящая работа начинается с атрибута `transform`. Он содержит два оператора, `translate` и `rotate`, и предназначен для изменения позиции начала координат `(0,0)` и направления осей X и Y. Мы сместим начало координат и повернем систему координат так, чтобы первая сторона сектора проходила вдоль оси X. (Поверьте, это упрощает вычисления.)

Итак, мы подошли к загадочному атрибуту `d`. Он содержит несколько команд графического вывода; в приведенном примере текущая точка смещается в позицию `(80,0)` (`M` означает «Move», т. е. «переместить»), после чего рисуется эллиптическая дуга (`A` означает «Arc», т. е. «дуга») с заданными свойствами. Наконец, мы рисуем линию (`L` означает «Line», то есть «линия») от текущей точки (конец дуги) до начала координат. Команда `Z` завершает фигуру линией, проведенной от текущей точки к начальной.

В свойствах команды `A`, если вас это действительно интересует, передаются две полуоси эллипса, величина угла поворота оси X в градусах, параметры `large-arc-flag` и `sweep-flag`, описывающие способ рисования дуги, и координаты X и Y конца дуги. В приведенном примере полуоси эллипса одинаковы (диаграмма должна быть круглой, а не эллиптической). Угол поворота оси X равен `0`. Далее идет флаг `large-arc-flag`, который равен `1`, если угловой размер сектора превышает 180 градусов; в противном случае он равен `0`. Параметр `sweep-flag` равен `0`, и последние два параметра, координаты X и Y, вычисляются. За подробной информацией об элементах `path` и `shape` обращайтесь к спецификации SVG.

8. Следующая задача – прорисовка всех сведений о магазинах. Для каждого сектора создается отдельная запись. Изначально все данные в области условных обозначений невидимы (`<g style="visibility:hidden;">` в терминологии SVG); перемещение указателя мыши над разными секторами приводит к отображению или скрытию разных сведений.

   При применении шаблона мы передаем несколько параметров, включая цвет образца и смещение сведений по оси Y. Шаблон вызывается один раз для каждого элемента `<store>`; это гарантирует, что в условных обозначениях будет представлен каждый сектор независимо от общего количества секторов. Для каждого сектора рисуется квадрат, заполненный соответствующим цветом, и рядом с ним выводится название магазина. Для каждой записи в условных обозначениях координата Y увеличивается на 20 пикселов. Чтобы сделать возможным вызов упоминавшихся ранее функций EcmaScript, мы

присваиваем каждой записи идентификатор: `legend1` для первого сектора, `legend2` для второго и т. д.

Шаблон выглядит так:

```
<xsl:template match="store" mode="legend">
  <xsl:param name="color" select="'red'"/>
  <xsl:param name="storeSales" select="'0'"/>
  <xsl:param name="y-legend-offset" select="'0'"/>
  <xsl:param name="position" select="'1'"/>

  <svg:g id="legend{$position}">

    <svg:text font-size="12px" text-anchor="start" x="240">
      <xsl:attribute name="y">
        <xsl:value-of select="$y-legend-offset"/>
      </xsl:attribute>
      <xsl:value-of select="name"/>
      <xsl:text>: </xsl:text>
      <xsl:value-of select="format-number($storeSales, '#,###.#')"/>
    </svg:text>

    <svg:path stroke="black" stroke-width="2px" fill="{$color}">
      <xsl:attribute name="d">
        <xsl:text>M 220 </xsl:text>
        <xsl:value-of select="$y-legend-offset - 10"/>
        <xsl:text> L 220 </xsl:text>
        <xsl:value-of select="$y-legend-offset"/>
        <xsl:text> L 230 </xsl:text>
        <xsl:value-of select="$y-legend-offset"/>
        <xsl:text> L 230 </xsl:text>
        <xsl:value-of select="$y-legend-offset - 10"/>
        <xsl:text> Z</xsl:text>
      </xsl:attribute>
    </svg:path>
  </svg:g>
</xsl:template>
```

На рис. 9.10 показано, как выглядит отдельная запись в области условных обозначений.

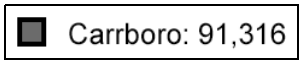


Рис. 9.10. *Запись в области условных обозначений*

И еще одно замечание: элемент группы SVG (`<svg:g>`) используется для группировки объектов. Каждая группа содержит текст и цветной квадрат, который показывает, какой сектор представляет тот или иной магазин. Включив эти данные в группу и присвоив ей имя, мы можем задать значение свойства `visibility` и одновременно отобразить или скрыть все объекты группы.

9. Осталось решить последнюю задачу: вывести подробные сведения по каждому магазину компании. Название магазина и его процент от общего объема продаж выводятся шрифтом, цвет которого соответствует цвету сектора диаграммы. Для каждого бренда, продаваемого этим магазином, выводится название бренда и объем продаж (см. рис. 9.11). По аналогии с шаблоном для вывода условных обозначений мы объединяем несколько объектов в группу SVG. Шаблон:

```
<xsl:template match="store" mode="details">
  <xsl:param name="color"/>
  <xsl:param name="y-legend-offset"/>
  <xsl:param name="storeSales"/>
  <xsl:param name="totalSales"/>

  <svg:g visibility="hidden" id="details{$position}">
    <svg:text font-size="16px" font-weight="bold" text-anchor="start"
      fill="black" x="220.5" y="{$y-legend-offset + .5}">
      <xsl:value-of select="name"/><xsl:text> Sales:</xsl:text>
    </svg:text>
    <svg:text font-size="16px" font-weight="bold" text-anchor="start"
      fill="{$color}" x="220" y="{$y-legend-offset}">
      <xsl:value-of select="name"/><xsl:text> Sales:</xsl:text>
    </svg:text>
    <xsl:for-each select="brand">
      <svg:text font-size="12px" text-anchor="start" x="220"
        y="{$y-legend-offset + (position() * 20)}">
        <xsl:value-of select="@name"/>
        <xsl:text>: </xsl:text>
        <xsl:value-of select="format-number(., '#,###.#')"/>
      </svg:text>
    </xsl:for-each>
    <svg:text font-size="14px" font-weight="bold" text-anchor="start"
      fill="black" x="220.5"
      y="{$y-legend-offset + 20.5 + (count(brand) * 20)}">
      <xsl:value-of
        select="format-number($storeSales div $totalSales, '##%')"/>
      <xsl:text> of total</xsl:text>
    </svg:text>
    <svg:text font-size="14px" font-weight="bold" text-anchor="start"
      fill="{$color}" x="220"
      y="{$y-legend-offset + 20 + (count(brand) * 20)}">
      <xsl:value-of
        select="format-number($storeSales div $totalSales, '##%')"/>
    </svg:text>
  </svg:g>
</xsl:template>
```

Обратите внимание: объект рисуется как невидимый (`visibility="hidden"`); для отображения или скрытия условных обозначений и подробных сведений будут использоваться эффекты JavaScript. В таблице стилей название текущего магазина выводится тем же

цветом, который использовался для рисования сектора; далее выводятся объемы продаж по каждому продукту, продаваемому в этом магазине, и процент от общих продаж. Сведения о продуктах выводятся со смещением в 20 пикселов, а последняя строка с процентом от общего объема продаж выводится на 20 пикселов ниже последнего бренда.

На рис. 9.11 показано, как выглядят подробные сведения по конкретному магазину.

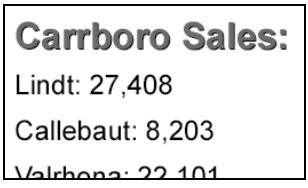


Рис. 9.11. *Подробные сведения по конкретному магазину*

И последний нюанс: название магазина и процент продаж выводятся дважды: черным цветом и цветом магазина. Черный текст смещается на 0,5 пиксела вниз и вправо от цветного текста, находящегося на переднем плане; так создается симпатичная имитация тени.

Полная таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- piechart.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  extension-element-prefixes="xalan-java saxon-java msxsl ora months"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:xalan-java="http://xml.apache.org/xslt/java"
  xmlns:saxon-java="java:java.lang.Math"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt"
  xmlns:ora="http://www.oreilly.com/xslt"
  xmlns:months="http://www.oreilly.com/xslt/months">

  <months:month sequence="1">January</months:month>
  <months:month sequence="2">February</months:month>
  <months:month sequence="3">March</months:month>
  <months:month sequence="4">April</months:month>
  <months:month sequence="5">May</months:month>
  <months:month sequence="6">June</months:month>
  <months:month sequence="7">July</months:month>
  <months:month sequence="8">August</months:month>
  <months:month sequence="9">September</months:month>
  <months:month sequence="10">October</months:month>
  <months:month sequence="11">November</months:month>
  <months:month sequence="12">December</months:month>

  <msxsl:script implements-prefix="ora" language="C#">
```

```
  <![CDATA[
    public double cos(double d)
    {
      return Math.Round(Math.Cos(d), 4);
    }

    public double sin(double d)
    {
      return Math.Round(Math.Sin(d), 4);
    }
  ]]>
</msxsl:script>

<xsl:output method="xml"/>

<xsl:variable name="totalSales" select="sum(//brand)"/>
<xsl:variable name="stores" select="count(/sales/store)"/>

<xsl:template match="/">
  <svg:svg width="450" height="300" version="1.1"
    baseProfile="full"
    xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:ev="http://www.w3.org/2001/xml-events">
    <xsl:comment>
      ECMAScript to handle mouse events
    </xsl:comment>

    <!-- Сгенерированный файл SVG включает три функции:            -->
    <!-- - mouse_over обычно выводит подробные сведения             -->
    <!--   по магазину вместо условных обозначений.                  -->
    <!-- - mouse_out обычно восстанавливает условные обозначения    -->
    <!--   при выходе указателя мыши за границы сектора.             -->
    <!-- - click обрабатывает щелчок кнопкой мыши на секторе.       -->
    <!--   Если это первый щелчок, сделанный на секторе,             -->
    <!--   то объемы продаж по соответствующему магазину             -->
    <!--   отображаются до тех пор, пока пользователь не щелкнет     -->
    <!--   на другом секторе круговой диаграммы.                     -->
    <!--   Если это второй щелчок, вывод подробных сведений         -->
    <!--   отключается и диаграмма возвращается                     -->
    <!--   к стандартному поведению мыши.                            -->

    <svg:script type="text/ecmascript">
      <xsl:text disable-output-escaping="yes">

  detailsViewOn = false;
  currentItem = 0;

  function mouse_over(selectedItem, totalItems)
  {
    if (!detailsViewOn || selectedItem == currentItem)
    {
      for (i = 1; totalItems >= i; i++)
      {
        obj = svgDocument.getElementById("details" + i);
        obj.setAttributeNS(null, "visibility", "hidden");
```

```
            obj = svgDocument.getElementById("legend" + i);
            obj.setAttributeNS(null, "visibility", "hidden");
          }

          obj = svgDocument.getElementById("details" + selectedItem);
          obj.setAttributeNS(null, "visibility", "visible");
        }
      }

      function mouse_out(totalItems)
      {
        if (!detailsViewOn)
        {
          for (i = 1; totalItems >= i; i++)
          {
            obj = svgDocument.getElementById("details" + i);
            obj.setAttributeNS(null, "visibility", "hidden");
            obj = svgDocument.getElementById("legend" + i);
            obj.setAttributeNS(null, "visibility", "visible");
          }
        }
      }

      function click(selectedItem, totalItems)
      {
        if (selectedItem != currentItem)
        {
          currentItem = selectedItem;
          mouse_over(selectedItem, totalItems);
          detailsViewOn = true;
        }
        else
        {
          obj = svgDocument.getElementById("details" + selectedItem);
          if (obj.getAttributeNS(null, "visibility") == "visible")
          {
            detailsViewOn = false;
            currentItem = 0;
          }
          else
          {
            currentItem = selectedItem;
            mouse_over(selectedItem, totalItems);
            detailsViewOn = true;
          }
        }
      }
          //      </xsl:text></svg:script>
        <xsl:text>
</xsl:text>

        <!-- Построение заголовка и подзаголовка.                -->
        <!-- Мы выводим заголовок и подзаголовок на диаграмме    -->
        <!-- и создаем элемент <svg:title> в самом SVG документе. -->
```

```
<xsl:variable name="titleText">
  <xsl:value-of select="/report/caption/heading"/>
  <xsl:text> - </xsl:text>
  <xsl:value-of
    select="document('')/*/months:month
            [@sequence=current()/report/@month]"/>
  <xsl:text> </xsl:text>

  <xsl:value-of select="/report/@year"/>
</xsl:variable>

<svg:title>
  <xsl:value-of select="$titleText"/>
</svg:title>
<svg:text font-size="24px" text-anchor="start"
  font-weight="bold" x="10" y="40">
  <xsl:value-of select="$titleText"/>
</svg:text>
<svg:text font-size="14px" text-anchor="middle" y="60" x="225">
  <xsl:value-of select="/report/caption/subheading"/>
</svg:text>

<xsl:variable name="totalSales" select="sum(//brand)"/>
<xsl:variable name="stores" select="count(/report/store)"/>

<xsl:for-each select="/report/store">
  <xsl:variable name="storeSales" select="sum(brand)"/>
  <xsl:variable name="color">
    <xsl:choose>
      <xsl:when test="(position() mod 6) = 1">
        <xsl:text>red</xsl:text>
      </xsl:when>
      <xsl:when test="(position() mod 6) = 2">
        <xsl:text>orange</xsl:text>
      </xsl:when>
      <xsl:when test="(position() mod 6) = 3">
        <xsl:text>purple</xsl:text>
      </xsl:when>
      <xsl:when test="(position() mod 6) = 4">
        <xsl:text>blue</xsl:text>
      </xsl:when>
      <xsl:when test="(position() mod 6) = 5">
        <xsl:text>green</xsl:text>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>gray</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>
  <xsl:variable name="explode" select="position()=1"/>

  <!-- Для каждого магазина на круговой диаграмме -->
  <!-- создается сектор                           -->
  <xsl:apply-templates select=".">
```

```
            <xsl:with-param name="color" select="$color"/>
            <xsl:with-param name="storeSales" select="$storeSales"/>
            <xsl:with-param name="totalSales" select="$totalSales"/>
            <xsl:with-param name="runningTotal"
              select="sum(preceding-sibling::store/brand)"/>
            <xsl:with-param name="stores" select="$stores"/>
            <xsl:with-param name="explode" select="$explode"/>
            <xsl:with-param name="position" select="position()"/>
          </xsl:apply-templates>

          <!-- Создание условного обозначения для каждого магазина -->
          <xsl:apply-templates select="." mode="legend">
            <xsl:with-param name="color" select="$color"/>
            <xsl:with-param name="storeSales" select="$storeSales"/>
            <xsl:with-param name="y-legend-offset"
              select="90 + (position() * 20)"/>
            <xsl:with-param name="position" select="position()"/>
          </xsl:apply-templates>

          <!-- Создание подробных сведений (изначально скрытых) -->
          <!-- для каждого магазина.                             -->
          <xsl:apply-templates select="." mode="details">
            <xsl:with-param name="color" select="$color"/>
            <xsl:with-param name="position" select="position()"/>
            <xsl:with-param name="y-legend-offset" select="110"/>
            <xsl:with-param name="storeSales" select="$storeSales"/>
            <xsl:with-param name="totalSales" select="$totalSales"/>
          </xsl:apply-templates>

        </xsl:for-each>
      </svg:svg>
    </xsl:template>

    <!-- Для каждого магазина создается сектор (<svg:path>)             -->
    <!-- и обработчики событий onmouseover(), onmouseout() и onclick(). -->
    <xsl:template match="store">
      <xsl:param name="color" select="'red'"/>
      <xsl:param name="runningTotal" select="'0'"/>
      <xsl:param name="totalSales" select="'0'"/>
      <xsl:param name="storeSales" select="'0'"/>
      <xsl:param name="stores" select="'5'"/>
      <xsl:param name="explode"/>
      <xsl:param name="position" select="'1'"/>

      <xsl:variable name="currentAngle">
        <xsl:choose>
          <xsl:when
            test="function-available('xalan-java:java.lang.Math.toRadians')">
            <xsl:value-of
              select="xalan-java:java.lang.Math.toRadians(
                      ($storeSales div $totalSales) * 360.0)"/>
          </xsl:when>
          <xsl:when
            test="function-available('saxon-java:toRadians')">
```

```
        <xsl:value-of
          select="saxon-java:toRadians(
                  ($storeSales div $totalSales) * 360.0)"/>
      </xsl:when>
      <xsl:when test="function-available('ora:cos')">
        <xsl:value-of
          select="($storeSales div $totalSales) *
                  6.283185307179586476925286766559"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:message terminate="yes">
           <xsl:text>Sorry, this stylesheet can't generate any </xsl:text>
          <xsl:text>useful results
without trigonometric </xsl:text>
          <xsl:text>functions available. </xsl:text>
        </xsl:message>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <svg:path fill="{$color}" stroke="black" stroke-width="2px"
    fillrule="evenodd" stroke-linejoin="bevel">
    <xsl:attribute name="onmouseout">
      <xsl:text>mouse_out(</xsl:text>
      <xsl:value-of select="$stores"/>
      <xsl:text>);</xsl:text>
    </xsl:attribute>
    <xsl:attribute name="transform">
      <xsl:choose>
        <xsl:when test="$explode">
          <xsl:text>translate(</xsl:text>
          <xsl:choose>
            <xsl:when
              test="function-available('xalan-java:java.lang.Math.cos')">
              <xsl:value-of
                select="(xalan-java:java.lang.Math.cos(
                        $currentAngle div 2) * 20) + 100"/>
              <xsl:text>,</xsl:text>
              <xsl:value-of
               select="(xalan-java:java.lang.Math.sin($currentAngle div 2)
                        * -20) + 160"/>
            </xsl:when>
            <xsl:when
              test="function-available('saxon-java:cos')">
              <xsl:value-of
                select="(saxon-java:cos($currentAngle div 2) * 20) + 100"/>
              <xsl:text>,</xsl:text>
              <xsl:value-of
               select="(saxon-java:sin($currentAngle div 2) * -20) + 160"/>
            </xsl:when>
            <xsl:when test="function-available('ora:cos')">
              <xsl:value-of
```

```
            select="(ora:cos($currentAngle div 2) * 20) + 100"/>
          <xsl:text>,</xsl:text>
          <xsl:value-of
            select="(ora:sin($currentAngle div 2) * -20) + 160"/>
        </xsl:when>
      </xsl:choose>
      <xsl:text>) </xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>translate(100,160) </xsl:text>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:text> rotate(</xsl:text>
 <xsl:value-of select="-1 * (($runningTotal div $totalSales) * 360.0)"/>
  <xsl:text>)</xsl:text>
</xsl:attribute>
<xsl:attribute name="onmouseover">
  <xsl:text>mouse_over(</xsl:text>
  <xsl:value-of select="$position"/>
  <xsl:text>, </xsl:text>
  <xsl:value-of select="$stores"/>
  <xsl:text>);</xsl:text>
</xsl:attribute>
<xsl:attribute name="onclick">
  <xsl:text>click(</xsl:text>
  <xsl:value-of select="$position"/>
  <xsl:text>, </xsl:text>
  <xsl:value-of select="$stores"/>
  <xsl:text>);</xsl:text>
</xsl:attribute>
<xsl:attribute name="d">
  <xsl:text>M 80 0 A 80 80 0 </xsl:text>
  <xsl:choose>
    <xsl:when test="$currentAngle > 3.14">
      <xsl:text>1 </xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>0 </xsl:text>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:text>0 </xsl:text>
  <xsl:choose>
   <xsl:when test="function-available('xalan-java:java.lang.Math.cos')">
      <xsl:value-of
        select="xalan-java:java.lang.Math.cos($currentAngle) * 80"/>
      <xsl:text> </xsl:text>
      <xsl:value-of
        select="xalan-java:java.lang.Math.sin($currentAngle) * -80"/>
    </xsl:when>
    <xsl:when test="function-available('saxon-java:cos')">
      <xsl:value-of select="saxon-java:cos($currentAngle) * 80"/>
      <xsl:text> </xsl:text>
```

```
          <xsl:value-of select="saxon-java:sin($currentAngle) * -80"/>
        </xsl:when>
        <xsl:when test="function-available('ora:cos')">
          <xsl:value-of
            select="ora:cos($currentAngle) * 80"/>
          <xsl:text> </xsl:text>
          <xsl:value-of
            select="ora:sin($currentAngle) * -80"/>
        </xsl:when>
      </xsl:choose>
      <xsl:text> L 0 0 Z </xsl:text>
    </xsl:attribute>
  </svg:path>
</xsl:template>

<!-- Создание записи для каждого магазина в области условных обозначений. -->
<xsl:template match="store" mode="legend">
  <xsl:param name="color" select="'red'"/>
  <xsl:param name="storeSales" select="'0'"/>
  <xsl:param name="y-legend-offset" select="'0'"/>
  <xsl:param name="position" select="'1'"/>
  <svg:g id="legend{$position}">

    <svg:text font-size="12px" text-anchor="start" x="240">
      <xsl:attribute name="y">
        <xsl:value-of select="$y-legend-offset"/>
      </xsl:attribute>
      <xsl:value-of select="name"/>
      <xsl:text>: </xsl:text>
      <xsl:value-of select="format-number($storeSales, '#,###.#')"/>
    </svg:text>

    <svg:path stroke="black" stroke-width="2px" fill="{$color}">
      <xsl:attribute name="d">
        <xsl:text>M 220 </xsl:text>
        <xsl:value-of select="$y-legend-offset - 10"/>
        <xsl:text> L 220 </xsl:text>
        <xsl:value-of select="$y-legend-offset"/>
        <xsl:text> L 230 </xsl:text>
        <xsl:value-of select="$y-legend-offset"/>
        <xsl:text> L 230 </xsl:text>
        <xsl:value-of select="$y-legend-offset - 10"/>
        <xsl:text> Z</xsl:text>
      </xsl:attribute>
    </svg:path>
  </svg:g>
</xsl:template>

<!-- Для каждого магазина создается скрытая запись -->
<!-- с подробными сведениями о продажах.           -->
<xsl:template match="store" mode="details">
  <xsl:param name="color" select="black"/>
  <xsl:param name="position" select="'0'"/>
```

```
    <xsl:param name="y-legend-offset"/>
    <xsl:param name="storeSales"/>
    <xsl:param name="totalSales"/>

    <svg:g visibility="hidden" id="details{$position}">
      <svg:text font-size="16px" font-weight="bold" text-anchor="start"
        fill="black" x="220.5">
        <xsl:attribute name="y">
          <xsl:value-of select="$y-legend-offset + .5"/>
        </xsl:attribute>
        <xsl:value-of select="name"/><xsl:text> Sales:</xsl:text>
      </svg:text>
      <svg:text font-size="16px" font-weight="bold" text-anchor="start"
        fill="{$color}" x="220">
        <xsl:attribute name="y">
          <xsl:value-of select="$y-legend-offset"/>
        </xsl:attribute>
        <xsl:value-of select="name"/><xsl:text> Sales:</xsl:text>
      </svg:text>

      <xsl:for-each select="brand">
        <svg:text font-size="12px" text-anchor="start" x="220">
          <xsl:attribute name="y">
            <xsl:value-of select="$y-legend-offset + (position() * 20)"/>
          </xsl:attribute>
          <xsl:value-of select="@name"/>
          <xsl:text>: </xsl:text>
          <xsl:value-of select="format-number(., '#,###.#')"/>
        </svg:text>
      </xsl:for-each>
      <svg:text font-size="14px" font-weight="bold" text-anchor="start"
        fill="black" x="220.5"
        y="{$y-legend-offset + 20.5 + (count(brand) * 20)}">
       <xsl:value-of select="format-number($storeSales div $totalSales, '##%')"/>
        <xsl:text> of total</xsl:text>
      </svg:text>
      <svg:text font-size="14px" font-weight="bold" text-anchor="start"
        fill="{$color}" x="220"
        y="{$y-legend-offset + 20 + (count(brand) * 20)}">
       <xsl:value-of select="format-number($storeSales div $totalSales, '##%')"/>

      </svg:text>
    </svg:g>
  </xsl:template>

</xsl:stylesheet>
```

В этом примере функции расширения XSLT используются для наделения процессора XSLT новыми возможностями. Нам понадобилась пара простых тригонометрических функций, и средства использования существующих классов Java в Saxon и Xalan упростили включение новых возможностей. Подобный механизм может использоваться для вызова статических методов классов Java в любой точке, где они вам понадобятся, причем для этого не придется писать ни одной строки кода Java.

Написание расширений на других языках

Одной из приятных особенностей механизма расширения Xalan-J является поддержка Bean Scripting Framework (BSF) – библиотеки с открытым исходным кодом от Apache Software Foundation, позволяющей выполнять код, написанный на различных сценарных языках. Мы возьмем рассмотренную ранее таблицу стилей SVG и реализуем ее снова, но на этот раз функции расширения будут записываться на других языках. Все подробности определения и вызова функций расширения BSF будут подробно описаны в первом примере; в остальных примерах мы ограничимся кратким выделением отличий.

Bean Scripting Framework поддерживает много языков, в том числе JRuby, Jython, Groovy, Jacl, NetRexx, PerlScript, Tcl и VBScript. Если вы работаете на платформе Microsoft, BSF также поддерживает Windows Script Technologies, так что ваш выбор в этом случае оказывается еще шире.

Для использования BSF необходимо включить в `CLASSPATH` два файла: *bsf.jar*, доступный по адресу *http://jakarta.apache.org/bsf/index.html*, и *commons-logging-1.1.jar*, доступный по адресу *http://jakarta.apache.org/commons/logging/*, а также JAR-файлы для используемого языка. Учтите, что системные требования могут измениться; например, предыдущие версии BSF не требовали включения компонента logging.

Jython

Наш обзор языков, поддерживаемых в BSF, начинается с Jython – реализации Python, написанной на Java. Естественно, для идентификации кода расширения в Xalan необходимо выполнить некоторые предварительные операции. Мы начнем с них, а затем перейдем к исходному коду функций расширения. Прежде всего необходимо определить префиксы используемых пространств имен:

```
<!-- piechart-jython.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:months="http://www.oreilly.com/xslt/months"
  xmlns:lxslt="http://xml.apache.org/xslt"
  xmlns:jython-extension="http://www.jython.org"
  extension-element-prefixes="jython-extension"
  exclude-result-prefixes="lxslt">
```

Мы генерируем разметку SVG, поэтому необходимо определить пространство имен `svg`. Пространство имен `months` предназначено для элементов `<months>`, которые будут использоваться в таблице отображения `document('')` (просто для разнообразия в этом случае мы обходимся без расширений EXSLT). Пространство имен `lxslt` используется Xalan для

обращения к Bean Scripting Framework, а префикс `jython-extension` идентифицирует функции расширения, написанные на Jython.

В данном примере префикс пространства имен `jython-extension` связывается с URI-адресом `http://www.jython.org`, соответствующим домашней странице проекта Jython. На самом деле этот URI-адрес может быть любым; как вы вскоре убедитесь, для BSF важно название языка.

Наш код размещается в элементе `<lxslt:component>`:

```
  <lxslt:component prefix="jython-extension"
    functions="cos sin">
    <lxslt:script lang="jython">
import math

def cos(d):
  return math.cos(d)

def sin(d):
  return math.sin(d)
    </lxslt:script>
  </lxslt:component>
```

Атрибут `prefix` элемента `<lxslt:component>` связывает код с префиксом пространства имен, а атрибут `functions` перечисляет функции расширения, предоставляемые этим кодом. Элемент `<lxslt:script>` содержит код расширения. *Атрибут* `lang` *элемента* `<lxslt:script>` *должен совпадать с названием языка, указанным в используемой реализации Bean Scripting Framework (с точностью до регистра символов)*. Если заменить значение атрибута `lang` строкой `Jython`, произойдет фатальная ошибка.

Теперь, когда код Jython должным образом упакован и помечен, остается лишь вызвать его:

```
<xsl:value-of
  select="(jython-extension:cos(number($currentAngle div 2)) * 20) + 100"/>
```

Если не считать расширения `jython-extension` перед вызовом функции, остальной код таблицы стилей остается неизменным. Функции `cos()` и `sin()` являются частью математической библиотеки Python.

Для использования функций расширения необходимо включить в `CLASSPATH` файл *jython.jar*, доступный по адресу *http://www.jython.org*, в дополнение к уже упоминавшимся файлам *bsf.jar* и *commons-logging-1.1.jar*.

Сейчас мы рассмотрим сценарный код и конфигурацию `CLASSPATH` для всех остальных языков.

JRuby

JRuby – это реализация популярного языка Ruby, написанная на Java. Мы определяем префикс пространства имен `jruby-extension` и связы-

ваем его с URI-адресом `http://jruby.codehaus.org`. Код таблицы стилей для JRuby выглядит так:

```
<?xml version="1.0"?>
<!-- piechart-jruby.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:months="http://www.oreilly.com/xslt/months"
  xmlns:lxslt="http://xml.apache.org/xslt"
  xmlns:jruby-extension="http://jruby.codehaus.org"
  extension-element-prefixes="jruby-extension"
  exclude-result-prefixes="lxslt">
...
  <lxslt:component prefix="jruby-extension"
    functions="cos sin">
    <lxslt:script lang="ruby">
      def cos(d)
        Math::cos(d)
      end

      def sin(d)
        Math::sin(d)
      end
    </lxslt:script>
  </lxslt:component>
```

Для использования функций расширения необходимо включить в `CLASSPATH` файл *jruby.jar*, доступный по адресу *http://jruby.codehaus.org*, в дополнение к уже упоминавшимся файлам *bsf.jar* и *commons-logging-1.1.jar*.

В этом случае атрибут `lang` должен содержать строку `ruby`. Попытка использования строк `JRuby`, `jruby` или `Ruby`, как бы логично они ни выглядели, приведет к ошибке времени выполнения.

JavaScript

Для демонстрации поддержки JavaScript будет использовано ядро Mozilla Rhino JavaScript. Мы определяем префикс пространства имен `javascript-extension` и связываем его с URI-адресом `http://www.mozilla.org/rhino`. Код таблицы стилей для JavaScript выглядит так:

```
<?xml version="1.0"?>
<!-- piechart-javascript.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:months="http://www.oreilly.com/xslt/months"
  xmlns:lxslt="http://xml.apache.org/xslt"
  xmlns:javascript-extension="http://www.mozilla.org/rhino"
  extension-element-prefixes="javascript-extension"
```

```
  exclude-result-prefixes="lxslt">
...
  <lxslt:component prefix="javascript-extension"
    functions="cos sin">
    <lxslt:script lang="javascript">
      function cos(d)
      {
        return Math.cos(d);
      }

      function sin(d)
      {
        return Math.sin(d);
      }
    </lxslt:script>
  </lxslt:component>
```

Для использования функций расширения необходимо включить в CLASSPATH файл *js.jar*, доступный по адресу *http://www.mozilla.org/rhino*, в дополнение к уже упоминавшимся файлам *bsf.jar* и *commons-logging-1.1.jar*.

Jacl

Jacl – это интерпретатор Tcl (Tool Command Language), написанный полностью на Java. Мы определяем префикс пространства имен jacl-extension и связываем его с URI-адресом http://tcljava.sourceforge.net. Код таблицы стилей выглядит так:

```
<?xml version="1.0"?>
<!-- piechart-jacl.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:months="http://www.oreilly.com/xslt/months"
  xmlns:lxslt="http://xml.apache.org/xslt"
  xmlns:jacl-extension="http://tcljava.sourceforge.net"
  extension-element-prefixes="jacl-extension"
  exclude-result-prefixes="lxslt">
...
  <lxslt:component prefix="jacl-extension"
    functions="cos sin">
    <lxslt:script lang="jacl">
      proc cos {d}  {expr cos($d)}
      proc sin {d}  {expr sin($d)}
    </lxslt:script>
  </lxslt:component>
```

Для использования функций расширения необходимо включить в CLASSPATH файлы *jacl.jar* и *tcljava.jar*, доступные по адресу *http://tcljava.sourceforge.net*, в дополнение к уже упоминавшимся файлам *bsf.jar* и *commons-logging-1.1.jar*.

Использование функций расширения из библиотеки EXSLT

Ранее уже упоминался проект EXSLT, целью которого является определение общего набора функций расширения XSLT. Для нашего следующего примера мы используем функции из библиотеки EXSLT. И Saxon, и Xalan имеют встроенную поддержку EXSLT, что значительно упрощает использование функций из библиотеки EXSLT.

EXSLT поддерживается многими процессорами XSLT, однако следует учитывать, что уровень поддержки EXSLT не однороден. Часть функций не поддерживается некоторыми процессорами, и не во всех процессорах функции обладают одинаковыми сигнатурами и результатами.

EXSLT предоставляет восемь категорий функций:

Общие

Общие функции для работы с типами данных и наборами узлов. Находятся в пространстве имен `http://exslt.org/common`.

Дата и время

Функции для работы с датой и временем. Находятся в пространстве имен `http://exslt.org/dates-and-times`.

Динамические

Функции динамической обработки выражений XPath. Находятся в пространстве имен `http://exslt.org/dynamic`.

Функции

Элементы расширения и функции для определения пользовательских функций. Находятся в пространстве имен `http://exslt.org/functions`.

Математические

Функции для тригонометрических вычислений, вычисления логарифмов, возведения в степень и других математических операций. Находятся в пространстве имен `http://exslt.org/math`.

Генератор случайных чисел

Единственная функция (`random-sequence()`), генерирующая последовательность случайных чисел в интервале от `0` до `1`. Находится в пространстве имен `http://exslt.org/random`.

Регулярные выражения

Функции для работы с регулярными выражениями. Находятся в пространстве имен `http://exslt.org/regular-expressions`.

Множества

Функции для вычисления разности и пересечения множеств. Находятся в пространстве имен `http://exslt.org/sets`.

Строки

Функции для выполнения операций со строками. Находятся в пространстве имен `http://exslt.org/strings`.

Многие функции и элементы EXSLT были включены в XSLT 2.0 как часть языка. Проект EXSLT объединяет наиболее востребованные возможности, отсутствующие в XSLT 1.0, поэтому неудивительно, что EXSLT оказывает существенное влияние на архитектуру и функциональность XSLT 2.0.

Также следует учитывать, что процессор XSLT, поддерживающий EXSLT, не обязан поддерживать все функции, определяемые в `http://exslt.org`. Между реализациями существуют некоторые различия; будьте внимательны.

Мы используем EXSLT в трех местах: при вызове функций `cos()` и `sin()` и при вызове функции `month-name()`, возвращающей название текущего месяца. Благодаря `month-name()` нам не приходится использовать элементы `<months:month>`, как это делалось ранее.

Чтобы не усложнять описание, мы рассмотрим только те части таблицы стилей EXSLT, которые отличаются от исходной таблицы стилей с построением круговой диаграммы. Прежде всего, как и следовало ожидать, элемент `<xsl:stylesheet>` объявляет другие пространства имен:

```
<?xml version="1.0"?>
<!-- piechart-exslt.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:math="http://exslt.org/math"
  xmlns:dates-and-times="http://exslt.org/dates-and-times"
  extension-element-prefixes="dates-and-times math">
```

В нашем примере проверяется доступность математических функций расширения и функций даты/времени из EXSLT. Функции EXSLT принадлежат двум разным пространствам имен, поэтому мы определяем префиксы `dates-and-times` и `math` в дополнение к применявшимся ранее.

Чтобы использовать функцию `month-name()`, придется немного потрудиться. Вот как выглядит код:

```
<xsl:choose>
  <xsl:when test="function-available('math:cos')">
    <xsl:variable name="dateString">
      <xsl:text>--</xsl:text>
      <xsl:if test="string-length(/report/@month) = 1">
        <xsl:text>0</xsl:text>
      </xsl:if>
      <xsl:value-of select="/report/@month"/>
      <xsl:text>--</xsl:text>
    </xsl:variable>
    <xsl:value-of
```

```
        select="dates-and-times:month-name($dateString)"/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="/report/@year"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="/report/@month"/>
      <xsl:text>/</xsl:text>
      <xsl:value-of select="/report/@year"/>
    </xsl:otherwise>
  </xsl:choose>
```

Если функция `dates-and-times:month-name()` доступна, мы используем ее для получения названия месяца, к которому относится отчет. Для получения названия месяца реализациям `month-name()`, входящим в поставку Xalan и Saxon, должна передаваться строка в формате `--MM--`. (Это единственный формат, который работает в обоих процессорах. Каждый процессор может поддерживать и другие форматы, но использование формата, общего для обоих процессоров, упростит таблицу стилей.) Мы создаем переменную `$dateString`, соответствующую этому формату; если значение атрибута `month` элемента `<report>` состоит из одного символа, мы ставим перед ним начальный нуль. Если функции EXSLT доступны, вызов `month-name('--08--')` вернет строку `August`. Если функции недоступны, таблица стилей использует атрибут `month` напрямую и создает заголовок вида `8/2006`.

Обратите внимание на проверку доступности функции `math:cos` вызовом `function-available()`. Вызов `function-available('dates-and-times:month-name')` в Xalan-J 2.7.0 возвращает `false`, даже если функция доступна. Проверка функции EXSLT `cos()` возвращает `true`, поэтому мы используем именно ее; для нас важен лишь сам факт поддержки EXSLT.

Другое отличие таблицы стилей EXSLT от исходной заключается в том, что поскольку EXSLT не определяет функцию `toRadians()`, нам приходится определять ее самостоятельно. Вот как выглядят версия EXSLT и версия расширения Saxon:

```
<xsl:when test="function-available('math:cos')">
  <xsl:value-of
    select="($storeSales div $totalSales) *
            6.283185307179586476925286766559"/>
</xsl:when>
<xsl:when
  test="function-available('saxon-java:toRadians')">
  <xsl:value-of
    select="saxon-java:toRadians(
            ($storeSales div $totalSales) * 360.0)"/>
</xsl:when>
```

Остальная часть таблицы стилей практически не изменилась. Например, координаты для функции SVG `translate()` генерируются следующим образом:

```
<xsl:when test="function-available('math:cos')">
  <xsl:value-of
    select="(math:cos($currentAngle div 2) * 20) + 100"/>
  <xsl:text>,</xsl:text>
  <xsl:value-of
    select="(math:sin($currentAngle div 2) * -20) + 160"/>
</xsl:when>
```

Обращение к базе данных из элемента расширения

В первое издание книги был включен отдельный элемент расширения, обращавшийся к базе данных SQL. С тех пор у процессоров Saxon и Xalan появились библиотеки расширения, решающие эту задачу за нас. В этом разделе я покажу, как пользоваться этими расширениями.

В следующих примерах используется СУБД Apache Derby, распространяемая с открытым исходным кодом (*http://db.apache.org/derby/*). Вот как выглядят команды Derby и команды SQL, создающие и заполняющие базу данных:

```
connect 'jdbc:derby://localhost:1527/books;create=true';
create schema doug;
set schema doug;
create table compbks (ISBN varchar(10) primary key, title varchar(50),
  author varchar(50), pages int, price double, publisher varchar(50));
insert into compbks values ('0596527217', 'XSLT', 'Doug Tidwell',
  800, 49.95, 'O''Reilly');
insert into compbks values ('0974152129', 'DocBook XSL: The Complete Guide',
  'Bob Strayton', 560, 49.95, 'Sagehill Enterprises');
insert into compbks values ('1565925807', 'DocBook: The Definitive Guide',
  'Norman Walsh and Leonard Muellner', 652, 39.95, 'O''Reilly');
insert into compbks values ('0596009747', 'XSLT Cookbook', 'Sal Mangano',
  751, 49.95, 'O''Reilly');
insert into compbks values ('0596003277', 'Learning XSLT',
  'Michael Fitzgerald', 352, 34.95, 'O''Reilly');
```

Мы создаем базу данных с именем `books` и таблицу `compbks`, после чего вставляем в таблицу пять записей.

Поддержка SQL в Saxon требует, чтобы имя пользователя соответствовало имени схемы базы данных; по этой причине мы создаем схему `doug` и связываем ее с базой данных. Кроме того, при обращении к базе данных должен использоваться пароль, содержащий не менее одного символа; вскоре мы рассмотрим эти вопросы более подробно.

Обращение к базе данных в Saxon

Процессор XSLT Saxon предоставляет набор элементов расширения для использования функциональности SQL. Существуют элементы

расширения для подключения к базе данных и отключения от нее, для выполнения запроса, для выполнения обновлений, вставки и удаления в таблицах. Чтобы избежать излишнего усложнения примера, мы применяем элемент `<sql:connection>` для подключения к базе данных, затем производим выборку данных при помощи элемента `<sql:query>` и закрываем подключение элементом `<sql:close>`.

Полный код таблицы стилей выглядит так:

```
<?xml version="1.0"?>
<!-- saxon-sql.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sql="java:/net.sf.saxon.sql.SQLElementFactory"
  xmlns:saxon="http://saxon.sf.net/"
  extension-element-prefixes="saxon sql">

  <xsl:output method="html"/>

  <xsl:template match="/">

    <!-- Создание подключения JDBC -->
    <xsl:variable name="connection"
      as="java:java.sql.Connection"
      xmlns:java="http://saxon.sf.net/java-type">
      <sql:connect database="jdbc:derby://localhost:1527/books"
        driver="org.apache.derby.jdbc.ClientDriver"
        user="doug"
        password="x"/>
    </xsl:variable>

    <!-- Выполнение запроса -->
    <xsl:variable name="queryResults">
      <sql:query connection="$connection" table="compbks"
        column="*" row-tag="tr" column-tag="td"/>
    </xsl:variable>

    <html>
      <head>
        <title>Computer books in our database</title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>Computer books in our database</h1>
        <p>Here are the
        <xsl:value-of select="count($queryResults/tr)"/>
        computer books we have in stock:</p>
        <table border="1" cellpadding="5">
          <tr>
            <th>ISBN</th>
            <th>Title</th>
            <th>Author</th>
            <th>Pages</th>
            <th>List price</th>
            <th>Publisher</th>
```

```
            </tr>
            <xsl:copy-of select="$queryResults"/>
          </table>
        </body>
      </html>

      <!-- Закрытие подключения -->
      <sql:close connection="$connection"/>
    </xsl:template>

</xsl:stylesheet>
```

Элементы расширения Saxon используются в трех местах. Сначала мы создаем подключение к базе данных элементом `<sql:connect>`. При этом необходимо указать URL-адрес базы данных, драйвер Java JDBC, который будет использоваться для обращения к ней, имя пользователя и пароль. Имя пользователя и пароль должны содержать хотя бы по одному символу. Даже если ваша база данных не защищена паролем (как база Derby в нашем примере), все равно необходимо задать имя и пароль. Ситуация дополнительно усложняется тем, что имя пользователя должно совпадать с именем схемы базы данных. Вот почему при создании базы данных Derby создается схема базы данных `doug`.

Второй элемент расширения, используемый в приведенном примере, – `<sql:qeuery>`. Именно здесь происходит фактическое выполнение запроса и получение итогового набора. Переменная `$queryResults` содержит дерево элементов, представляющее итоговый набор. В нашем примере задаются имя таблицы базы данных, возвращаемые столбцы (наш пример означает `select *`, хотя при желании можно указать более конкретную выборку) и элементы, используемые для представления записей и полей в результатах. Мы генерируем таблицу HTML, поэтому используем элементы `tr` и `td`.

После использования элемента `<sql:query>` переменная `$queryResults` содержит дерево элементов, представляющее результаты запроса к базе данных. Мы используем функцию `count()` для вывода количества книг, найденных по запросу, затем создаем таблицу. Элементы HTML `<table>`, `<tr>` и `<th>` создают структуру таблицы, после чего содержимое переменной `$queryResults` копируется в выходной поток.

После построения таблицы HTML подключение к базе данных закрывается элементом `<sql:close>`.

Результат применения таблицы стилей показан на рис. 9.12.

Убедитесь в том, что переменная `CLASSPATH` содержит классы пакета расширения Saxon SQL и драйверов JDBC; в противном случае таблицы стилей работать не будут.

Элементы расширения SQL в Saxon построены на базе JDBC, поэтому они отличаются исключительной гибкостью. Например, чтобы ис-

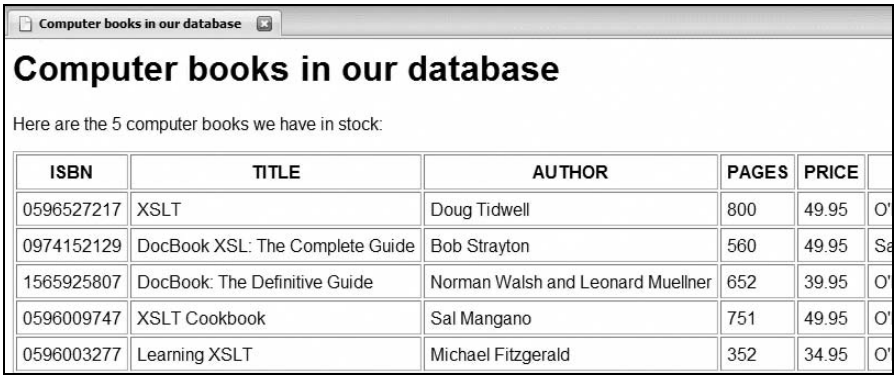


***Рис. 9.12.** Таблица HTML, построенная с помощью элементов расширения SQL в Saxon*

пользовать эту таблицу стилей с DB2, достаточно изменить атрибуты элемента `<sql:connect>`:

```
<?xml version="1.0"?>
<!-- saxon-sql2.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sql="java:/net.sf.saxon.sql.SQLElementFactory"
  xmlns:saxon="http://saxon.sf.net/"
  extension-element-prefixes="saxon sql">

  ...

    <!-- Создание подключения JDBC -->
    <xsl:variable name="connection"
      as="java:java.sql.Connection"
      xmlns:java="http://saxon.sf.net/java-type">
      <sql:connect database="jdbc:db2:books"
        driver="COM.ibm.db2.jdbc.app.DB2Driver"
        user="Skippy"
        password="xxxxxxxx"/>
    </xsl:variable>

  ...

</xsl:stylesheet>
```

Версия базы данных для DB2 защищена паролем, поэтому мы передаем имя пользователя и пароль, определенные в системе. Изменился всего один элемент, а таблица стилей работает с совершенно другим источником данных. Как и при работе с любой базой данных, необходимо знать класс драйвера JDBC и формат URL-адреса базы, но это единственное, что будет отличать две таблицы стилей. Результаты применения новой таблицы стилей в точности совпадают с предыдущими.

В имени класса драйвера JDBC учитывается регистр символов. Если вместо `COM.ibm.db2...` запросить класс `com.ibm.db2...`, произойдет ошибка времени выполнения.

Обращение к базе данных в Xalan

Xalan предоставляет функции расширения для обращения к произвольным источникам данных JDBC. В следующем примере функция `sql:new()` создает подключение к базе данных, функция `sql:query()` выполняет запрос, а функция `sql:close()` закрывает подключение. Кроме того, мы используем функцию `sql:disableStreamingMode()`, чтобы обойти одну «особенность» SQL-библиотеки Xalan.

Таблица стилей:

```
<?xml version="1.0"?>
<!-- xalan-sql.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sql="org.apache.xalan.lib.sql.XConnection"
  extension-element-prefixes="sql">

  <xsl:output method="html"/>

  <xsl:template match="/">

    <!-- Создание подключения JDBC -->
    <xsl:variable name="books"
      select="sql:new('org.apache.derby.jdbc.ClientDriver',
              'jdbc:derby://localhost:1527/books')"/>

    <!-- Обходное решение для устранения ошибки в расширении SQL Xalan? -->
    <xsl:variable name="streaming"
      select="sql:disableStreamingMode($books)"/>

    <!-- Выполнение запроса -->
    <xsl:variable name="queryResults"
      select="sql:query($books, 'select * from doug.compbks')"/>

    <html>
      <head>
        <title>Computer books in our database</title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>Computer books in our database</h1>
        <p>Here are the
        <xsl:value-of select="count($queryResults/sql/row-set/row)"/>
        computer books we have in stock:</p>
        <table border="1" cellpadding="5">
          <tr>
            <xsl:for-each select="$queryResults/sql/metadata/column-header">
              <th>
                <xsl:value-of select="@column-label"/>
              </th>
```

```
            </xsl:for-each>
          </tr>
          <xsl:apply-templates select="$queryResults/sql/row-set/row"/>
        </table>
      </body>
    </html>

    <!-- Close the connection -->
    <xsl:value-of select="sql:close($books)"/>
  </xsl:template>

  <xsl:template match="row">
    <tr>
      <xsl:apply-templates select="col"/>
    </tr>
  </xsl:template>

  <xsl:template match="col">
    <td>
      <xsl:value-of select="text()"/>
    </td>
  </xsl:template>

</xsl:stylesheet>
```

Сначала создается переменная `$books`, представляющая подключение к базе данных. Обратите внимание: SQL-библиотека Xalan не требует передачи имени пользователя и пароля; мы просто задаем класс драйвера JDBC и URL-адрес базы данных. Вызов `sql:disableStreamingMode($books)` позволяет получить доступ ко всем результатам в итоговом наборе. Без него итоговый набор всегда содержит только одну запись. Xalan-J 2.7.0 работает именно так, хотя возможно, будущие версии SQL-библиотеки Xalan будут работать более осмысленно.

Обратите внимание: библиотека Xalan открывает доступ к информации, предоставляемой классом `ResultSetMetaData`. В данном примере используются заголовки столбцов нашей базы данных, определяемые в иерархии XPath `sql/metadata/column-header`. Каждый заголовок столбца отображается в таблице в виде элемента `<th>`. SQL-библиотека Saxon не предоставляет этой информации.

Также следует отметить, что Xalan предоставляет большую гибкость в формулировке запроса. При желании в команду можно включить секцию `ORDER BY` или `GROUP BY`. Имя таблицы также задается иначе: мы запрашиваем таблицу `doug.compbks`, то есть таблицу с именем `compbks`, находящуюся в схеме базы данных `doug`. Если просто запросить таблицу `compbks`, драйвер JDBC сообщит, что эта таблица не существует. (Запомните, для работы таблицы стилей в Saxon необходимо добавить схему `doug`.)

После генерирования заголовков столбцов таблицы мы переходим к обработке данных итогового набора. Строки и столбцы итогового набора представлены элементом `<row-set>`; каждая строка соответствует эле-

менту <row> внутри <row-set>, а каждый столбец – элементу <col> внутри элемента <row>. Шаблоны, определяемые для <row> и <col>, достаточно прямолинейны. Результаты в точности совпадают с результатами таблиц стилей, в которых использовались SQL-расширения Saxon.

В завершение этого короткого обзора SQL-расширений Xalan мы рассмотрим вторую таблицу стилей, работающую с базой данных DB2. Отличия, как и следовало ожидать, весьма незначительны:

```
<?xml version="1.0"?>
<!-- xalan-sql2.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sql="org.apache.xalan.lib.sql.XConnection"
  extension-element-prefixes="sql">

  ...

    <!-- Создание подключения JDBC -->
    <xsl:variable name="books"
      select="sql:new('COM.ibm.db2.jdbc.app.DB2Driver',
               'jdbc:db2:books', 'Skippy', 'xxxxxxxx')"/>

    ...

    <!-- Выполнение запроса -->
    <xsl:variable name="queryResults"
      select="sql:query($books, 'select * from compbks')"/>

  ...

</xsl:stylesheet>
```

Мы должны задать другой класс драйвера JDBC и URL-адрес базы данных в другом формате. Эта версия sql:new() получает два параметра с именем пользователя и паролем. Другое отличие заключается в том, что для таблицы базы данных не указывается схема. В DB2 по умолчанию в качестве имени схемы используется имя пользователя, создавшего базу данных, поэтому достаточно запросить таблицу compbks. Если таблица была создана с другой схемой, имя этой схемы должно быть задано явно.

Создание фотоальбома с помощью элементов расширения

В последнем примере мы создадим элемент расширения, который генерирует таблицу HTML со всеми фотографиями из заданного каталога. Элемент расширения получает три параметра: каталог, количество изображений в каждой строке таблицы и флаг вывода имени файла под фотографией. Сгенерированная таблица содержит уменьшенные версии всех фотографий; каждое миниатюрное изображение представ-

ляет собой ссылку на полноразмерный файл. На рис. 9.13 показан примерный вид сгенерированной страницы.

Мы рассмотрим три версии расширения. Первые две написаны на Java (для Xalan и для Saxon). Эти два процессора используют разные механизмы обращения к программному коду; кроме того, в них используются разные механизмы возвращения дерева элементов HTML таблице стилей. Третья версия написана на C# для платформы .NET. Версия для .NET оформлена в виде *функции* расширения, а не элемента, хотя она также возвращает дерево нужных элементов HTML. HTML-код, сгенерированный тремя реализациями, не различается.

Все версии нашего расширения используют следующий исходный файл XML:

```
<?xml version="1.0"?>
<!-- photo-album.xml -->
<photo-album
  directory="c:\photos"
  imagesPerRow="5"
  includeFilenames="yes"/>
```

Исходный код XML содержит три параметра, необходимых для нашего расширения: каталог с фотографиями, количество изображений в строке и флаг включения имен файлов в таблицу. Как вы увидите, код расширения в этих трех примерах значительно различается, но входные данные и вывод HTML одинаковы.

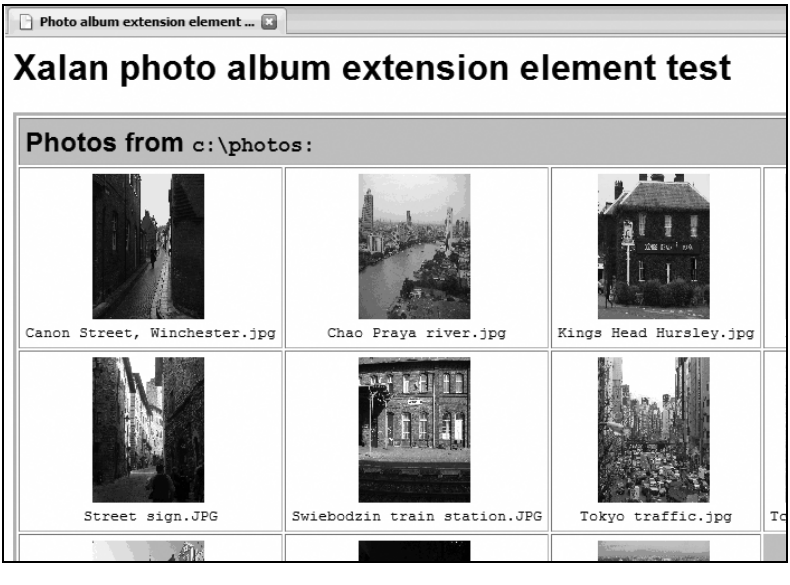


Рис. 9.13. *Фотоальбом, сгенерированный с помощью элемента расширения*

Общая логика кода тоже остается неизменной:

1. Получить атрибуты от элемента расширения. В одних случаях это делается в таблице стилей, в других элемент расширения получает собственные параметры.
2. Проверить, существует ли заданный каталог. Если каталог не существует, создать таблицу с соответствующим сообщением.
3. Если каталог существует, проверить, содержит ли он графические файлы. Если таких файлов нет, создать таблицу с соответствующим сообщением.
4. Если каталог существует и содержит графические файлы, создать таблицу с элементами `<img>`, содержащими ссылки на эти файлы.

Java-версия для Xalan

Начнем с Java-версии расширения для Xalan. Возвращаемое значение элемента расширения относится к типу `org.w3c.dom.Element`. Прежде всего рассмотрим управляющую таблицу стилей XSLT:

```
<?xml version="1.0"?>
<!-- xalan-photo-album.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xpa="xalan://com.oreilly.xslt.xalan.XalanPhotoAlbumExtension"
  extension-element-prefixes="xpa">

  <xsl:output method="html"/>

  <xsl:template match="photo-album">
    <html>
      <head>
        <title>Photo album extension element test</title>
      </head>
      <body style="font-family: sans-serif;">
        <xsl:choose>
          <xsl:when test="element-available('xpa:XalanPhotoAlbum')">
            <h1>Xalan photo album extension element test</h1>
            <xpa:XalanPhotoAlbum/>
          </xsl:when>
          <xsl:otherwise>
            <p>
              <i>[Sorry, the photo album function is not available.]</i>
            </p>
          </xsl:otherwise>
        </xsl:choose>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Код элемента расширения:

```
/*
 * XalanPhotoAlbumExtension.java
 * Created on Nov 28, 2006 by Doug Tidwell
 */

package com.oreilly.xslt.xalan;

import java.io.File;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.apache.xalan.extensions.XSLProcessorContext;
import org.apache.xalan.templates.ElemExtensionCall;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

// Класс создает фотоальбом со всеми графическими
// изображениями из заданного каталога.
public class XalanPhotoAlbumExtension
{
  // XalanPhotoAlbum - имя элемента расширения. Обратите внимание:
  // элемент возвращает узел Element, который вставляется в выходное дерево.
  public static Element XalanPhotoAlbum(XSLProcessorContext context,
      ElemExtensionCall elem)
    throws ParserConfigurationException
  {
    Element contextNode = (Element) context.getContextNode();

    // В параметрах передается имя каталога, количество
    // изображений в строке, а также флаг вывода имен файлов.

    int imagesPerRowValue;
    try
    {
      imagesPerRowValue =
        Integer.parseInt(contextNode.getAttribute("imagesPerRow"));
    }
    catch (NumberFormatException nfe)
    {
      imagesPerRowValue = 5;
    }

    String directoryName = contextNode.getAttribute("directory");

    boolean includeFilenamesFlag =
      (contextNode.getAttribute("includeFilenames").
          equalsIgnoreCase("yes"));

    // Элемент строит дерево DOM, поэтому элемент документа
    // и некоторые другие элементы создаются заранее.
    // Возвращается элемент <table> с графикой
    // или сообщением об ошибке, если возникнут проблемы.
```

```
 DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
 DocumentBuilder db = dbf.newDocumentBuilder();
 Document doc = db.newDocument();

Element a = null, br = null, img = null, span = null,
 table = null, td = null, tr = null;

 table = doc.createElement("table");
 table.setAttribute("border", "3");
 table.setAttribute("cellpadding", "5");

 File dir = new File(directoryName);

 // Если каталог не существует, вернуть сообщение об ошибке.
 if (!dir.exists())
 {
   tr = doc.createElement("tr");

   td = doc.createElement("td");
   td.setAttribute("style", "font-weight:bold; font-size: 150%");

   td.appendChild(doc.createTextNode("The directory "));

   span = doc.createElement("span");
   span.setAttribute("style", "font-family: monospace;");
   span.appendChild(doc.createTextNode(directoryName));
   td.appendChild(span);

   td.appendChild(doc.createTextNode(" does not exist!"));
   tr.appendChild(td);
   table.appendChild(tr);
 }
 else
 {
   // Использовать фильтр имен файлов для поиска графики.
   String[] graphicsFiles = dir.list(new GraphicsFilenameFilter());
   int numFiles = 0;
   if (graphicsFiles != null)
     numFiles = graphicsFiles.length;

   // Вернуть сообщение об ошибке, если в каталоге нет графических файлов.
   if (numFiles == 0)
   {
     tr = doc.createElement("tr");

     td = doc.createElement("td");
     td.setAttribute("style", "font-weight:bold; font-size: 150%");

     td.appendChild(doc.createTextNode("The directory "));

     span = doc.createElement("span");
     span.setAttribute("style", "font-family: monospace;");
     span.appendChild(doc.createTextNode(directoryName));
     td.appendChild(span);

     td.appendChild(doc.createTextNode(" doesn't contain any images!"));
     tr.appendChild(td);
```

```
  table.appendChild(tr);
}

// Графические изображения получены, обрабатываем...
else
{
  tr = doc.createElement("tr");

  td = doc.createElement("td");
  td.setAttribute("colspan", String.valueOf(imagesPerRowValue));
  td.setAttribute("style",
                  "font-weight:bold; background: #CCCCCC; " +
                  "font-size: 150%");
  td.appendChild(doc.createTextNode("Photos from "));

  span = doc.createElement("span");
  span.setAttribute("style", "font-family: monospace;");
  span.appendChild(doc.createTextNode(directoryName + ":"));
  td.appendChild(span);

  tr.appendChild(td);
  table.appendChild(tr);

  int filesProcessed = 0;
  boolean emptyColumnNotCreated = true;

  while (filesProcessed < numFiles)
  {
    tr = doc.createElement("tr");
    for (int i = 0; i < imagesPerRowValue; i++)
    {
      if (filesProcessed < numFiles)
      {
        td = doc.createElement("td");
        td.setAttribute("style", "text-align: center;");

        String qualifiedFilename = "file:\\\\" +
        directoryName + File.separatorChar +
        graphicsFiles[filesProcessed];

        a = doc.createElement("a");
        a.setAttribute("href", qualifiedFilename);

        img = doc.createElement("img");
        img.setAttribute("src", qualifiedFilename);
        img.setAttribute("width", "100px");
        img.setAttribute("height", "130px");
        img.setAttribute("border", "0");
        a.appendChild(img);
        td.appendChild(a);

        if (includeFilenamesFlag)
        {
          br = doc.createElement("br");
          td.appendChild(br);

          span = doc.createElement("span");
```

```
                span.setAttribute("style", "font-family: monospace;");
                span.appendChild(doc.createTextNode
                                 (graphicsFiles[filesProcessed]));
                td.appendChild(span);
              }
              tr.appendChild(td);
              filesProcessed++;
            }
            // Все файлы перечислены, создаем пустой столбец
            else if (emptyColumnNotCreated)
            {
              td = doc.createElement("td");
              td.setAttribute("colspan",
                              String.valueOf(imagesPerRowValue - i));
              td.setAttribute("bgcolor", "#CCCCCC");
              tr.appendChild(td);
              emptyColumnNotCreated = false;
            }
          }
          table.appendChild(tr);
        }
      }
    }
    return table;
  }
}
```

Элемент расширения получает от Xalan два аргумента: `org.apache.xalan.extensions.XSLTProcessorContext` и `org.apache.xalan.templates.ElemExtensionCall`. Объект `XSLTProcessorContext` используется для обращения к элементу `<photo-album>` в исходном документе XML. Xalan и Saxon предоставляют аналогичные контекстные объекты элементам и функциям расширения, хотя они обладают разной структурой, методами и возможностями.

Если вам доводилось писать код Java для работы с моделью DOM, вы узнаете основные приемы. Сначала создается элемент DOM с именем `table`, затем элемент DOM с именем `tr`, далее элементы DOM с именем `td`, после чего в них помещаются элементы `text` или `img` и т. д. Код получается монотонным, что типично для кода DOM, но зато он тривиален.

Например, при создании элемента `td` мы создаем элемент `a` и задаем значения его атрибутов. Затем создается элемент `img`, для которого также задаются значения атрибутов. Построенный элемент `img` включается в элемент `a`, после чего элемент `a` включается в элемент `td`. Если в исходном коде XML указано, что в таблицу HTML должны быть включены имена файлов, мы создаем текстовый узел с именем файла и добавляем его в элемент `td`. Когда элемент `td` будет завершен, он включается в элемент `tr`; завершенный элемент `tr` включается в элемент `table`. В конце кода мы получаем завершенный элемент `table`, ко-

торый возвращается процессору XSLT. Процессор использует элемент и всех его потомков в выходном документе.

Для упрощения кода Java мы используем объект Java `FilenameFilter`, который отбирает файлы по маске. Элемент расширения вызывает этот класс в следующей строке кода Java:

```
String[] graphicsFiles = dir.list(new GraphicsFilenameFilter());
```

Если файл не является графическим, его не нужно включать в альбом. Код фильтра выглядит так:

```
/*
 * GraphicsFilenameFilter.java
 * Created on Nov 28, 2006 by Doug Tidwell
 */

package com.oreilly.xslt.xalan;
// Фильтр для Saxon выглядит так же, если не считать
// имени пакета (com.oreilly.xslt.saxon)

import java.io.File;
import java.io.FilenameFilter;

public class GraphicsFilenameFilter implements FilenameFilter
{

  public boolean accept(File dir, String name)
  {
    String lcName = name.toLowerCase();
    if (lcName.endsWith(".jpg")  ||
        lcName.endsWith(".jpeg") ||
        lcName.endsWith(".png")  ||
        lcName.endsWith(".gif")  ||
        lcName.endsWith(".bmp"))
      return true;
    else
      return false;
  }

}
```

Файлы с расширениями `.jpg`, `.jpeg`, `.png`, `.gif` и `.bmp` успешно проходят через фильтр. Файлы, не соответствующие условию фильтрации, не включаются в список имен файлов. Фильтр используется при создании списка обрабатываемых файлов:

```
String[] graphicsFiles = dir.list(new GraphicsFilenameFilter());
```

Предполагается, что любой файл с соответствующим расширением содержит графические данные. Трудолюбивые читатели могут самостоятельно добавить код, который открывает каждый файл и убеждается в том, что он действительно содержит графические данные. Возможно, также будет полезно проверить размеры изображения и создать миниатюру, сохраняющую пропорции исходного изображения. Наконец, файлы JPEG могут содержать метаданные о дате и времени съем-

ки; отображение этой информации в выходных данных HTML – еще одно полезное дополнение к нашей программе.

Java-версия для Saxon

Элемент расширения для Saxon существенно сложнее Xalan-версии. Нам необходимо создать класс, реализующий интерфейс `net.sf.saxon.style.ExtensionElementFactory`. Этот класс отвечает за обработку атрибутов элемента `<SaxonPhotoAlbum>`, включая обработку шаблонов значений атрибутов в элементе. Следующий короткий файл решает эту задачу:

```
/*
 * PhotoAlbumElementFactory.java
 * Created on Nov 28, 2006 by Doug Tidwell
 */

package com.oreilly.xslt.saxon;

import net.sf.saxon.style.ExtensionElementFactory;

public class PhotoAlbumElementFactory
  implements ExtensionElementFactory
{
  public Class getExtensionClass(String elementName)
  {
    if (elementName.equals("SaxonPhotoAlbum"))
      return SaxonPhotoAlbum.class;
    return null;
  }
}
```

Код возвращает объект `class` для класса `SaxonPhotoAlbum`. Saxon использует полученный объект для создания объекта `SaxonPhotoAlbum`, обрабатывающего элемент в таблице стилей.

```
/*
 * SaxonPhotoAlbum.java
 * Created on Nov 28, 2006 by Doug Tidwell
 */

package com.oreilly.xslt.saxon;

import java.io.File;

import net.sf.saxon.event.Receiver;
import net.sf.saxon.event.ReceiverOptions;
import net.sf.saxon.expr.Expression;

import net.sf.saxon.expr.SimpleExpression;
import net.sf.saxon.expr.XPathContext;
import net.sf.saxon.instruct.Executable;
import net.sf.saxon.om.NamePool;
import net.sf.saxon.style.ExtensionInstruction;
import net.sf.saxon.style.StandardNames;
```

```
import net.sf.saxon.trans.XPathException;

public class SaxonPhotoAlbum
  extends ExtensionInstruction
{
  Expression directory;
  Expression imagesPerRow;
  Expression includeFilenames;

  @Override
  public void prepareAttributes() throws XPathException
  {
    String directoryAttr = attributeList.getValue("", "directory");
    if (directoryAttr == null)
    {
      compileError("The directory attribute is required.", "");
      directoryAttr = "";
    }
    directory = makeAttributeValueTemplate(directoryAttr);

    String imagesPerRowAttr =
      attributeList.getValue("", "imagesPerRow");
    if (imagesPerRowAttr == null)
      imagesPerRowAttr = "5";
    imagesPerRow = makeAttributeValueTemplate(imagesPerRowAttr);

    String includeFilenamesAttr =
      attributeList.getValue("", "includeFilenames");
    if (includeFilenamesAttr == null)
      includeFilenamesAttr = "no";
    includeFilenames = makeAttributeValueTemplate(includeFilenamesAttr);
  }

  @Override
  public Expression compile(Executable arg0) throws XPathException
  {
    SaxonPhotoAlbumInstruction spa =
      new SaxonPhotoAlbumInstruction(directory,
                                     imagesPerRow,
                                     includeFilenames);
    return spa;
  }

  private static class SaxonPhotoAlbumInstruction
    extends SimpleExpression {

    private static final long serialVersionUID = 1184239671886575198L;
    public static final int DIRECTORY = 0;
    public static final int IMAGES_PER_ROW = 1;
    public static final int INCLUDE_FILENAMES = 2;

    public SaxonPhotoAlbumInstruction(Expression directory,
        Expression imagesPerRow, Expression includeFilenames)
    {
      Expression[] subs = {directory, imagesPerRow, includeFilenames};
```

```
  setArguments(subs);
}

/**
 * Субкласс должен предоставлять один из методов evaluateItem(),
 * iterate() или process(). Этот метод указывает,
 * какой из трех методов предоставляется.
 */

public int getImplementationMethod()
{
    return Expression.PROCESS_METHOD;
}

public String getExpressionType()
{
    return "spa:SaxonPhotoAlbum";
}

public void process(XPathContext context) throws XPathException
{
  String directoryName =
    arguments[DIRECTORY].evaluateAsString(context);

  int imagesPerRowValue = 5;
  try
  {
    imagesPerRowValue =
        Integer.parseInt(arguments[IMAGES_PER_ROW].
                         evaluateAsString(context));
  }
  catch (NumberFormatException nfe)
  {
    imagesPerRowValue = 5;
  }
  if (!(imagesPerRowValue > 0))
  {
    imagesPerRowValue = 5;
  }

  boolean includeFilenamesFlag =
    arguments[INCLUDE_FILENAMES].evaluateAsString(context)
    .equals("yes");

  NamePool pool = context.getController().getNamePool();
  int anchorCode = pool.allocate("", "", "a");
  int borderCode = pool.allocate("", "", "border");
  int brCode = pool.allocate("", "", "br");
  int cellpaddingCode = pool.allocate("", "", "cellpadding");
  int colspanCode = pool.allocate("", "", "colspan");
  int heightCode = pool.allocate("", "", "height");
  int hrefCode = pool.allocate("", "", "href");
  int imgCode = pool.allocate("", "", "img");
  int spanCode = pool.allocate("", "", "span");
  int srcCode = pool.allocate("", "", "src");
```

```
int styleCode = pool.allocate("", "", "style");
int tableCode = pool.allocate("", "", "table");
int colCode = pool.allocate("", "", "td");
int rowCode = pool.allocate("", "", "tr");
int widthCode = pool.allocate("", "", "width");

Receiver out = context.getReceiver();

// Начало <table>
out.startElement(tableCode, StandardNames.XDT_UNTYPED, locationId, 0);
out.attribute(borderCode, StandardNames.XDT_UNTYPED,
              "3", locationId, 0);
out.attribute(cellpaddingCode, StandardNames.XDT_UNTYPED,
              "5", locationId, 0);

String[] graphicsFiles;
int numFiles = 0;

File dir = new File(directoryName);

// Если каталог не существует, вернуть сообщение об ошибке
if (!dir.exists())
{
  // Начало <tr>
  out.startElement(rowCode, StandardNames.XDT_UNTYPED, locationId, 0);

  // Начало <td>
  out.startElement(colCode, StandardNames.XDT_UNTYPED, locationId, 0);
  out.attribute(styleCode, StandardNames.XDT_UNTYPED,
                "font-weight: bold; font-size: 150%;", locationId, 0);
  out.characters("The directory ", locationId, 0);

  // Начало <span>
 out.startElement(spanCode, StandardNames.XDT_UNTYPED, locationId, 0);
  out.attribute(styleCode, StandardNames.XDT_UNTYPED,
                "font-family: monospace;", locationId, 0);
  out.characters(directoryName, locationId, 0);
  out.endElement(); // конец <span>

  out.characters(" does not exist!", locationId, 0);
  out.endElement(); // конец <td>

  out.endElement(); // конец <tr>
}
else
{
  // Использование фильтра для отбора графических файлов
  graphicsFiles = dir.list(new GraphicsFilenameFilter());
  if (graphicsFiles != null)
    numFiles = graphicsFiles.length;

  // Если каталог не содержит графических файлов,
  // вернуть сообщение об ошибке.
  if (numFiles == 0)
  {
    // Начало <tr>
```

```
  out.startElement(rowCode, StandardNames.XDT_UNTYPED, locationId, 0);
   // Начало <td>
  out.startElement(colCode, StandardNames.XDT_UNTYPED, locationId, 0);
  out.attribute(styleCode, StandardNames.XDT_UNTYPED,
                "font-weight: bold; background: #CCCCCC; " +
                "font-size: 150%;",
                locationId, 0);
  out.characters("The directory ", locationId, 0);
   // Начало <span>
  out.startElement(spanCode, StandardNames.XDT_UNTYPED, locationId, 0);
  out.attribute(styleCode, StandardNames.XDT_UNTYPED,
                "font-family: monospace;", locationId, 0);
  out.characters(directoryName, locationId, 0);
  out.endElement(); // конец <span>
  out.characters(" doesn't contain any images!", locationId, 0);

  out.endElement(); // конец <td>

  out.endElement(); // конец <tr>
 }

 // Графические изображения получены, обрабатываем...
 else
 {
   // Начало <tr>
  out.startElement(rowCode, StandardNames.XDT_UNTYPED, locationId, 0);

   // Начало <td>
  out.startElement(colCode, StandardNames.XDT_UNTYPED, locationId, 0);
  out.attribute(colspanCode, StandardNames.XDT_UNTYPED, String
      .valueOf(imagesPerRowValue), locationId, 0);
  out.attribute(styleCode, StandardNames.XDT_UNTYPED,
                "font-weight: bold; background: #CCCCCC; "
                    + "font-size: 150%;", locationId, 0);
  out.characters("Photos from ", locationId, 0);

   // Начало <span>
  out.startElement(spanCode, StandardNames.XDT_UNTYPED, locationId, 0);
  out.attribute(styleCode, StandardNames.XDT_UNTYPED,
                "font-family: monospace;", locationId, 0);
  out.characters(directoryName + ":", locationId, 0);
  out.endElement(); // конец <span>

  out.endElement(); // конец <td>

  out.endElement(); // конец <tr>
 }

int filesProcessed = 0;
boolean emptyColumnNotCreated = true;

while (filesProcessed < numFiles)
{
  // Начало <tr>
  out.startElement(rowCode, StandardNames.XDT_UNTYPED, locationId, 0);
```

```
for (int i = 0; i < imagesPerRowValue; i++)
{
  if (filesProcessed < numFiles)
  {
    // Начало <td>
   out.startElement(colCode, StandardNames.XDT_UNTYPED, locationId, 0);
   out.attribute(styleCode, StandardNames.XDT_UNTYPED,
                 "text-align: center;", locationId, 0);

   String qualifiedFilename = "file:\\\\" + directoryName
        + File.separatorChar + graphicsFiles[filesProcessed];

    // Начало <a>
   out.startElement(anchorCode, StandardNames.XDT_UNTYPED,
                    locationId, 0);
   out.attribute(hrefCode, StandardNames.XDT_UNTYPED,
                 qualifiedFilename, locationId, 0);

    // Начало <img>
   out.startElement(imgCode, StandardNames.XDT_UNTYPED, locationId, 0);
   out.attribute(widthCode, StandardNames.XDT_UNTYPED, "100px",
                 locationId, 0);
   out.attribute(heightCode, StandardNames.XDT_UNTYPED, "130px",
                 locationId, 0);
   out.attribute(srcCode, StandardNames.XDT_UNTYPED,
                 qualifiedFilename, locationId, 0);
   out.attribute(borderCode, StandardNames.XDT_UNTYPED, "0",
                 locationId, 0);
   out.endElement(); // конец <img>

   out.endElement(); // конец <a>

   if (includeFilenamesFlag)
   {
      // Начало <br>
    out.startElement(brCode, StandardNames.XDT_UNTYPED, locationId, 0);
    out.endElement(); // конец <br>

      // Начало <span style="font-family: monospace">
    out.startElement(spanCode, StandardNames.XDT_UNTYPED,
                     locationId, 0);
    out.attribute(styleCode, StandardNames.XDT_UNTYPED,
                  "font-family: monospace;", locationId, 0);
    out.characters(graphicsFiles[filesProcessed], locationId, 0);
       out.endElement(); // конец <span>
     }
     out.endElement(); // конец <td>
     filesProcessed++;
   }
   // Все файлы перечислены, создаем пустой столбец
   else if (emptyColumnNotCreated)
   {
     // Начало <td>
     out.startElement(colCode, StandardNames.XDT_UNTYPED,
                      locationId, 0);
```

```
                out.attribute(styleCode, StandardNames.XDT_UNTYPED,
                              "background: #CCCCCC;", locationId, 0);
                out.attribute(colspanCode, StandardNames.XDT_UNTYPED,
                              String.valueOf(imagesPerRowValue - i),
                              locationId, 0);
                out.characters(" ", locationId,
                               ReceiverOptions.DISABLE_ESCAPING);
                out.endElement(); // конец <td>
                emptyColumnNotCreated = false;
              }
            }
            out.endElement(); // конец <tr>
          }
        }
        out.endElement(); // конец <table>
      }
    }
  }
```

По аналогии с расширением Xalan Saxon предоставляет контекстный объект – в данном случае `net.sf.saxon.expr.XPathContext`. Расширение Saxon работает аналогично расширению Xalan, создавая элементы и атрибуты и возвращая их в виде древовидной структуры. Впрочем, существуют и различия; в самом начале необходимо определить объект Saxon `NamePool`:

```
NamePool pool = context.getController().getNamePool();
int anchorCode = pool.allocate("", "", "a");
int borderCode = pool.allocate("", "", "border");
int brCode = pool.allocate("", "", "br");
int cellpaddingCode = pool.allocate("", "", "cellpadding");
...
```

Мы определяем имена создаваемых элементов и атрибутов. Затем объект `out` используется для создания различных элементов таблицы. Объект `out` представляет собой объект Saxon `net.sf.saxon.event.Receiver`, получаемый от `XPathContext`. Объект `out` используется для создания элементов, атрибутов и текстовых узлов. Мы используем четыре метода класса `Receiver`: `startElement()`, `attribute()`, `characters()` и `endElement()`. Обратите внимание: метод `endElement()` вызывается без параметров; он просто завершает текущий создаваемый элемент. Комментарии в коде обозначают начало и конец элементов.

Как упоминалось ранее для Xalan-версии этого расширения, объект `FilenameFilter` отбирает из каталога графические файлы.

Версия для .NET

Версия фотоальбома для .NET несколько отличается от предыдущих. Расширение реализуется в виде *функции*, а не элемента расширения. Эта реализация выбрана для демонстрации альтернативного способа создания узлов в выходном дереве.

Наша функция расширения создает дерево DOM со всеми элементами HTML, необходимыми для выходного дерева. Версии этого элемента расширения для Xalan и Saxon заменяют элемент расширения в таблице стилей соответствующими сгенерированными узлами. В среде .NET функция расширения возвращает дерево DOM, включаемое в дерево результата.

Далее приводится код функции расширения. Он работает по тем же базовым принципам, что и другие версии расширения: сначала мы используем системные функции для получения списка файлов, а затем создаем элементы DOM для таблицы HTML:

```
/*
 * PhotoAlbum.cs
 * Created on Nov 28, 2006 by Doug Tidwell
 */

using System;
using System.IO;
using System.Text;
using System.Xml;

// Класс создает фотоальбом со всеми графическими
// изображениями из заданного каталога.
namespace com.oreilly.xslt
{
  class PhotoAlbum
  {
    // getPhotoListing - имя функции расширения.
    // Обратите внимание на то, что функция возвращает XMLNode;
    // этот узел вставляется в выходное дерево.
    public XmlNode getPhotoListing(String dirName,
                                   String imgPerRow,
                                   String incFilenames)
    {
      // В параметрах передается имя каталога, количество
      // изображений в строке, а также флаг вывода имен файлов.
      int imagesPerRow = int.Parse(imgPerRow);
      Boolean includeFilenames = (incFilenames.Equals("yes"));

      // Мы строим дерево DOM, поэтому
      // некоторые элементы создаются заранее.
      XmlElement anchorEl, breaklineEl, imageEl, spanEl,
          tableEl, tableDataEl, tableRowEl;
      XmlNode textNode;

      XmlDocument doc = new XmlDocument();
      XmlDocumentFragment df = doc.CreateDocumentFragment();
      tableEl = doc.CreateElement("table");

      tableEl.SetAttribute("border", "3");
      tableEl.SetAttribute("cellpadding", "5");

      Boolean directoryExists = true;
```

```
System.Collections.ArrayList allGraphicsFiles =
    new System.Collections.ArrayList();

// Добавляются все файлы с расширениями JPEG, PNG, GIF и BMP
// из заданного каталога. В Windows регистр символов в именах
// не учитывается, поэтому код получается немного проще,
// чем в Java-версии.
try
{
  allGraphicsFiles.InsertRange(allGraphicsFiles.Count,
      System.IO.Directory.GetFiles(dirName, "*.jpg"));
  allGraphicsFiles.InsertRange(allGraphicsFiles.Count,
      System.IO.Directory.GetFiles(dirName, "*.jpeg"));
  allGraphicsFiles.InsertRange(allGraphicsFiles.Count,
      System.IO.Directory.GetFiles(dirName, "*.png"));
  allGraphicsFiles.InsertRange(allGraphicsFiles.Count,
      System.IO.Directory.GetFiles(dirName, "*.gif"));
  allGraphicsFiles.InsertRange(allGraphicsFiles.Count,
      System.IO.Directory.GetFiles(dirName, "*.bmp"));
}

// Если каталог не существует, вернуть сообщение об ошибке.
catch (System.IO.DirectoryNotFoundException dnfe)
{
  directoryExists = false;
  tableRowEl = doc.CreateElement("tr");
  tableDataEl = doc.CreateElement("td");
  tableDataEl.SetAttribute("style", "font-weight: bold; " +
      "font-size: 150%;");
  textNode = doc.CreateTextNode("The directory ");
  tableDataEl.AppendChild(textNode);
  spanEl = doc.CreateElement("span");
  spanEl.SetAttribute("style", "font-family: monospace;");
  spanEl.AppendChild(doc.CreateTextNode(dirName));
  tableDataEl.AppendChild(spanEl);
  textNode = doc.CreateTextNode(" doesn't exist!");
  tableDataEl.AppendChild(textNode);
  tableRowEl.AppendChild(tableDataEl);
  tableEl.AppendChild(tableRowEl);
  df.AppendChild(tableEl);
}

int numFiles = allGraphicsFiles.Count;

if (directoryExists)
{
 // Вернуть сообщение об ошибке, если в каталоге нет графических файлов.
  if (numFiles == 0)
  {
    tableRowEl = doc.CreateElement("tr");
    tableDataEl = doc.CreateElement("td");
    tableDataEl.SetAttribute("style", "font-weight: bold; " +
                             "font-size: 150%;");
    textNode = doc.CreateTextNode("The directory ");
```

```
    tableDataEl.AppendChild(textNode);
    spanEl = doc.CreateElement("span");
    spanEl.SetAttribute("style", "font-family: monospace;");
    spanEl.AppendChild(doc.CreateTextNode(dirName));
    tableDataEl.AppendChild(spanEl);
    textNode = doc.CreateTextNode(" doesn't contain any images!");
    tableDataEl.AppendChild(textNode);
    tableRowEl.AppendChild(tableDataEl);
    tableEl.AppendChild(tableRowEl);
    df.AppendChild(tableEl);
  }

// Графические изображения получены, обрабатываем...
  else
  {
    // Сначала создается строка таблицы для заголовка.
    tableRowEl = doc.CreateElement("tr");
    tableDataEl = doc.CreateElement("td");
    tableDataEl.SetAttribute("colspan", imgPerRow);
    tableDataEl.SetAttribute("style", "font-weight: bold; " +
                             "background: #CCCCCC; font-size: 150%;");
    tableDataEl.AppendChild(doc.CreateTextNode("Photos from "));
    spanEl = doc.CreateElement("span");
    spanEl.SetAttribute("style", "font-family: monospace;");
    spanEl.AppendChild(doc.CreateTextNode(dirName + ":"));
    tableDataEl.AppendChild(spanEl);
    tableRowEl.AppendChild(tableDataEl);
    tableEl.AppendChild(tableRowEl);

    // Переходим к обработке файлов. Флаг emptyColumnNotCreated
    // сообщает, была ли создана пустая ячейка. Если в строке выводится
    // 5 изображений, а общее количество изображений равно 47,
    // последняя строка содержит пустую ячейку с атрибутом colspan = 3.
    // Пустая ячейка создается только один раз.
    int filesProcessed = 0;
    Boolean emptyColumnNotCreated = true;

    // Перебор всех файлов
    while (filesProcessed < numFiles)
    {
      tableRowEl = doc.CreateElement("tr");

      // Обработка следующих {imagesPerRow} изображений
      for (int i = 0; i < imagesPerRow; i++)
      {
        if (filesProcessed < numFiles)
        {
          tableDataEl = doc.CreateElement("td");
          tableDataEl.SetAttribute("style", "text-align: center;");
          String qualifiedFilename = "file:\\\\" +
              allGraphicsFiles[filesProcessed];
          anchorEl = doc.CreateElement("a");
          imageEl = doc.CreateElement("img");
          imageEl.SetAttribute("src", qualifiedFilename);
```

```
                imageEl.SetAttribute("width", "100px");
                imageEl.SetAttribute("height", "130px");
                imageEl.SetAttribute("border", "0");
                anchorEl.AppendChild(imageEl);
                tableDataEl.AppendChild(anchorEl);

            if (includeFilenames)
                {
                  breaklineEl = doc.CreateElement("br");
                  tableDataEl.AppendChild(breaklineEl);

                // Используем класс FileInfo для получения имени файла без пути.
                  FileInfo fi =
                    new FileInfo((String)allGraphicsFiles[filesProcessed]);
                  spanEl = doc.CreateElement("span");
                  spanEl.SetAttribute("style", "font-family: monospace;");
                  spanEl.AppendChild(doc.CreateTextNode(fi.Name));
                  tableDataEl.AppendChild(spanEl);
                }
                tableRowEl.AppendChild(tableDataEl);
                filesProcessed++;
              }

              // Если мы обработали все файлы и не создали
              // пустой столбец, сделаем это сейчас.
              else if (emptyColumnNotCreated)
              {
                tableDataEl = doc.CreateElement("td");
                Int32 colspan = (imagesPerRow - i);
                tableDataEl.SetAttribute("colspan", colspan.ToString());
                tableDataEl.SetAttribute("style", "background: #CCCCCC;");
                tableRowEl.AppendChild(tableDataEl);
                emptyColumnNotCreated = false;
              }
            }

            // Этот фрагмент создает элементы и присоединяет
            // их к соответствующему родительскому элементу.
            tableEl.AppendChild(tableRowEl);
          }
          df.AppendChild(tableEl);
        }
      }

      // После завершения обработки мы получаем набор узлов, содержащий
      // все необходимые элементы HTML. Он возвращается вызывающей
      // стороне, которая вставляет эти узлы в выходной поток.
      return df;
    }
  }
}
```

Код DOM в большей степени похож на расширение Xalan, нежели на расширение Saxon. В этой версии используются классы .NET, однако

общая структура кода достаточно близка к двум расширениям Java. Мы собираем имена файлов, создаем таблицу, создаем ячейку для каждого изображения и возвращаем элемент таблицы. В данном случае возвращается объект .NET `XmlDocumentFragment`.

Итак, функция расширения определена; теперь нам понадобится таблица стилей, в которой эта функция вызывается. Как упоминалось ранее, для расширения необходимо определить пространство имен:

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:ora="http://www.oreilly.com/xslt"
  extension-element-prefixes="ora">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Photo album extension function test</title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>
          <xsl:text>.Net Photo album extension function test</xsl:text>
        </h1>
        <xsl:variable name="album"
                      select="ora:getPhotoListing(/photo-album/@directory,
                      /photo-album/@imagesPerRow,
                      /photo-album/@includeFilenames)"/>
        <xsl:copy-of select="$album"/>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Так как реализация .NET представляет собой функцию расширения, мы присваиваем ее значение переменной. После создания переменной (фрагмента дерева результата) элемент `<xsl:copy-of>` копирует все содержимое переменной в выходное дерево.

Как и в более ранних примерах .NET, нам понадобится файл C#, создающий объект `XslCompiledTransformation`. Мы также создадим объект `PhotoAlbum` и свяжем его с префиксом пространства имен, определенным в таблице стилей. Остается лишь создать объект `XsltArgumentList`, связывающий все компоненты воедино.

```
/*
 * Main.cs
 * Created Nov 22, 2006 by Doug Tidwell
 */

using System;
```

```
using System.Text;
using System.Xml;
using System.Xml.Xsl;

namespace com.oreilly.xslt
{
  class MainClass
  {
    static void Main(string[] args)
    {
      // Создание объекта таблицы стилей и объекта XMLWriter,
      // который записывает выходные данные в файл.
      XslCompiledTransform stylesheet = new XslCompiledTransform();
      XmlTextWriter xWriter =
          new XmlTextWriter(args[2], Encoding.UTF8);

      // Использование объекта XsltSettings, который разрешает
      // выполнение сценариев (необходимо для расширений), а затем
      // загружает таблицу стилей.
      XsltSettings settings = new XsltSettings(true, true);
      stylesheet.Load(args[1], settings, new XmlUrlResolver());

      // Создание объекта расширения
      PhotoAlbum pa = new PhotoAlbum();

      // Включение объекта PhotoAlbum как объекта расширения
      // в XsltArgumentList.
      XsltArgumentList argList = new XsltArgumentList();
      argList.AddExtensionObject("http://www.oreilly.com/xslt", pa);

      // Когда подготовка будет завершена, вызываем метод Transform(),
      // который и выполняет всю работу...
      stylesheet.Transform(args[0], argList, xWriter);
    }
  }
}
```

Обратите внимание: в таблице стилей определяется пространство имен `http://www.oreilly.com/xslt` — то же, которое используется при создании `XlstArgumentList` в коде C#. Если два пространства не совпадают, процессор XSLT для .NET не сможет найти класс расширения.

Если все будет сделано верно, процессор сгенерирует страницу, показанную на рис. 9.14.

Большинство цифровых фотокамер создает файлы в формате JPEG, содержащие теги EXIF (Exchangeable Image Format). Теги содержат информацию о модели камеры, дате и времени съемки и т. д. Если вы хотите работать со всей информацией, доступной в JPEG-файлах с тегами EXIF, обращайтесь к описанию библиотеки EXIF Tag Parsing Library: *http://sourceforge.net/projects/libexif*. Доработка расширения и отображение этих технических данных в таблице (или временном окне при наведении указателя мыши на изображение) станет хорошим дополнением к нашему коду.

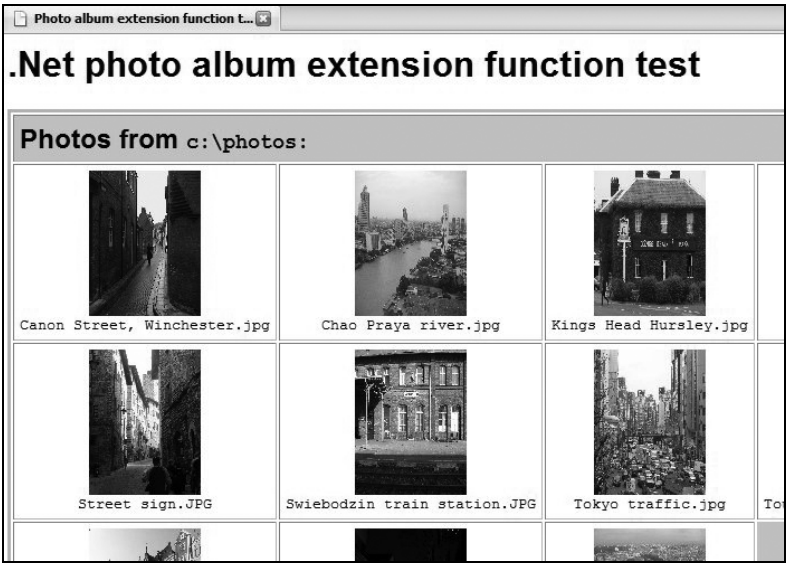


Рис. 9.14. *Фотоальбом, сгенерированный функцией расширения .NET*

Итоги

В этой главе была представлена подборка функций и элементов расширения, позволяющих включить в таблицу стилей нетривиальные вычислительные возможности. Мы генерировали выходные файлы, динамически создавали графику JPEG, создавали интерактивные изображения, взаимодействовали с базами данных и генерировали файлы HTML, созданные на основании данных из файловой системы.

Что касается собственно XSLT, мы создавали как элементы, так и функции расширения, возвращающие простые значения или деревья объектов DOM. Но что еще важнее, в этой главе были продемонстрированы разнообразные механизмы взаимодействия кода расширения с исходными документами XML для разных платформ, языков и процессоров XSLT. Если вы их освоите, не останется почти ничего, что бы вы не могли сделать в своих таблицах стилей.

A

Справочник по элементам XSLT

Данная глава представляет собой полный справочник по всем элементам, определенным в спецификации XSLT. Чтобы примеры оставались по возможности несложными, большинство из них генерирует простой текст вместо кода XML, HTML или XHTML. Это поможет нам сосредоточиться на XSLT, а не на генерировании свойств CSS, таблиц HTML или других деталей. Конечно, там, где примеры с XML, HTML или XHTML оказываются более содержательными, мы используем их, но большинство таблиц стилей по возможности упрощено.

[2.0] Атрибуты, общие для всех элементов XSLT

В любом элементе XSLT могут использоваться шесть стандартных атрибутов; мы определим их здесь, вместо того чтобы многократно переопределять их для каждого элемента XSLT. Хотя эти атрибуты могут использоваться с любым элементом XSLT, обычно они встречаются в таких элементах, как `<xsl:stylesheet>` и `<xsl:template>`.

Атрибуты

`version`

Определяет версию XSLT, используемую для обработки данного элемента. Этот атрибут может быть полезен, если некоторый элемент XSLT должен обрабатываться по правилам конкретной версии стандарта. Например, элемент `<xsl:value-of select="1 div 0"/>` по-разному работает в XSLT 1.0 и 2.0. В XSLT 2.0 он приводит к фатальной ошибке, а XSLT 1.0 возвращает бесконечность (`Infinity`). Использование записи `<xsl:value-of version="1.0" ...>` гарантирует, что будет применена обработка по правилам версии 1.0.

Хотя элемент `<xsl:output>` тоже поддерживает атрибут `version`, он задает значение атрибута `version` в выходных данных. Например, элемент `<xsl:output method="xml" version="1.1">` создает итоговый документ с объявлением XML `<?xml version="1.1" ...?>`.

exclude-result-prefixes

Перечисляет префиксы пространств имен, которые не должны копироваться в выходные данные. Обычно используется только с элементом <xsl:stylesheet>.

extension-element-prefixes

Определяет префиксы пространств имен, идентифицирующие элементы расширения. Процессор XSLT использует эти пространства имен для идентификации кода, обеспечивающего дополнительную обработку. Обычно используется только с элементом <xsl:stylesheet>.

xpath-default-namespace

Определяет пространство имен по умолчанию, используемое в выражениях и шаблонах XPath. Если преобразуемый документ использует пространство имен по умолчанию (например, http://www.oreilly.com), вы должны уточнить пространства имен всех элементов в документе. Определение xpath-default-namespace="http://www.oreilly.com" сообщает XPath, что пространство имен должно использоваться по умолчанию для элементов и имен типов в выражениях XPath. Если в разных частях преобразуемых документов используются разные пространства имен, вы можете использовать этот атрибут для смены пространства имен XPath по умолчанию. Обычно используется только с элементом <xsl:stylesheet>.

default-collation

Серия разделенных пробелами URI-адресов, определяющих схему упорядочения по умолчанию. Последовательность упорядочения по умолчанию используется элементами <xsl:key> и <xsl:for-each-group>, но она не влияет на упорядочение, используемое <xsl:sort>. Способ определения последовательностей упорядочения зависит от реализации; обращайтесь к документации своего процессора.

use-when

Определяет условие с использованием системного свойства, которое должно быть истинным для обработки элемента XSLT. Например, вызов функции system-property('xsl:is-schema-aware') позволит проверить, является ли процессор XSLT схемосовместимым.

[2.0] <xsl:analyze-string>

Позволяет проанализировать строку с применением регулярного выражения.

Изменения в XSLT 2.0

Элемент <xsl:analyze-string> появился в XSLT 2.0.

Категория

Инструкция.

Обязательные атрибуты

select

Выражение XPath, определяющее анализируемую строку. При необходимости выражение преобразуется в строку.

`regex`

Регулярное выражение. В регулярных выражениях часто используются фигурные скобки (`{` и `}`), которые XSLT интерпретирует как начало и конец шаблона значения атрибута. По этой причине все фигурные скобки в регулярных выражениях должны удваиваться. Например, регулярное выражение `"[0-9]{{5}}"` совпадает с числом из пяти цифр, а `"[0-9]{5}"` – с одной цифрой, за которой следует цифра `5`.

Совпадение регулярного выражения со строкой нулевой длины приводит к фатальной ошибке. Подробнее см. в приложении Е.

Необязательные атрибуты

`flags`

Атрибут `flags` управляет обработкой регулярного выражения. Поддерживаются четыре флага:

`s` Регулярное выражение обрабатывается в режиме «совпадения точки со всеми символами». При наличии этого флага метасимвол «точка» (`.`) совпадает с любым символом. При стандартной обработке (без флага `s`) оператор «точка» совпадает с любым символом, *кроме* символа новой строки (`
`). Этот флаг особенно полезен при обработке строк, содержащих внутренние символы новой строки.

`m` Регулярное выражение обрабатывается в многострочном режиме. В режиме по умолчанию метасимвол `^` совпадает с началом всей строки, а метасимвол `$` совпадает с концом каждой строки. В многострочном режиме `^` совпадает с началом, а `$` – с концом любой логической строки внутри «физической» строки.

`i` Регулярное выражение обрабатывается без учета регистра символов. Регулярное выражение `"a"` совпадает как с `"a"`, так и с `"A"`.

Учтите, что Юникод может создать массу проблем с этим флагом. Например, в спецификации «XQuery 1.0 and XPath 2.0 Functions and Operators» приводится пример с обозначением в Юникоде градусов по Кельвину (`K`), соответствующим букве `"K"`. Комбинация `regex="k"` и `flags="i"` совпадает как со знаком градусов по Кельвину, так и с буквами `"k"` (`k`) и `"K"` (`K`).

Другие символы Юникода не преобразуются в буквы. Например, символ римской цифры `I` (`ࡰ`) внешне похож на букву `I`, но не преобразуется в нее. К счастью, эти проблемы выходят за рамки данной книги.

`x` Перед поиском совпадения из регулярного выражения удаляются все символы пропусков (`	`, `
`, `
` и ` `). Другими словами, с флагом `x` регулярные выражения `"John Smith"` и `"JohnSmith"` эквивалентны. Этот флаг особенно удобен при разбиении длинного регулярного выражения на несколько строк, для того чтобы его было легче читать.

Флаги могут комбинироваться в любом порядке. Наборы параметров `'xis'` и `'six'` работают абсолютно одинаково.

Содержимое

Элемент <xsl:analyze-string> может содержать один необязательный элемент <xsl:matching-substring> и/или один необязательный элемент <xsl:non-matching-substring>. Если присутствуют оба элемента <xsl:matching-substring> и <xsl:non-matching-substring>, элемент <xsl:matching-substring> должен находиться на первом месте. Иначе говоря, все три элемента <xsl:analyze-string> в следующем листинге действительны:

```
<xsl:analyze-string select="." regex="a">
  <xsl:matching-substring>...</xsl:matching-substring>
</xsl:analyze-string>

<xsl:analyze-string select="." regex="a">
  <xsl:non-matching-substring>...</xsl:non-matching-substring>
</xsl:analyze-string>

<xsl:analyze-string select="." regex="a">
  <xsl:matching-substring>...</xsl:matching-substring>
  <xsl:non-matching-substring>...</xsl:non-matching-substring>
</xsl:analyze-string>
```

Элемент <xsl:analyze-string> может также содержать произвольное количество элементов <xsl:fallback>. Если ваш процессор XSLT 1.0 поддерживает режим прямой совместимости, элемент <xsl:fallback> позволяет определить реакцию процессора в случае, если элемент <xsl:analyze-string> не поддерживается. Например, таблица стилей может иметь следующую структуру:

```
<xsl:analyze-string...>
  <xsl:matching-substring>...</xsl:matching-substring>
  <xsl:fallback>
    <!-- Инструкции для процессора 1.0 -->
  </xsl:fallback>
</xsl:analyze-string>
```

Процессор XSLT 1.0 с прямой совместимостью обнаруживает элемент <xsl:analyze-string>, который он не может обработать. Каждый раз, когда процессор обнаруживает в пространстве имен xsl: неподдерживаемый элемент, он ищет внутри него элемент <xsl:fallback>. Если такой элемент будет обнаружен, процессор пытается обработать содержимое <xsl:fallback>.

Местонахождение

Элемент <xsl:analyze-string> находится в шаблоне.

Где определяется

XSLT, раздел 15 «Regular Expressions».

Поддержка регулярных выражений в XSLT 2.0, XPath 2.0 и XQuery 1.0 базируется на синтаксисе, определяемом в стандарте XML Schema. За полным описанием синтаксиса обращайтесь к приложению E и разделу «Regular Expressions» в «XML Schema Part 2: Datatypes». Спецификация «XQuery 1.0 and XPath 2.0 Functions and Operators» определяет расширения синтаксиса регулярных выражений XML Schema в разделе 7.6 «String Functions that Use Pattern Matching».

Пример

В следующем примере использования элемента `<xsl:analyze-string>` регулярное выражение преобразует телефонные номера в формате `999-999-9999` в формат `+1 (999) 999-9999`:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- analyze-string1.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:for-each select="phonelist/phonenumber">
      <xsl:analyze-string select="."
        regex="([0-9]{{3}})-(\p{{Nd}}{{3}})-([0-9]{{4}})">
        <xsl:matching-substring>
          <xsl:text>&#xA;+1 (</xsl:text>
          <xsl:value-of select="regex-group(1)"/>
          <xsl:text>) </xsl:text>
          <xsl:value-of select="regex-group(2)"/>
          <xsl:text>-</xsl:text>
          <xsl:value-of select="regex-group(3)"/>
        </xsl:matching-substring>
        <xsl:non-matching-substring>
          <xsl:text>&#xA;   Unrecognized phone number: </xsl:text>
          <xsl:value-of select="."/>
        </xsl:non-matching-substring>
      </xsl:analyze-string>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

В таблице стилей используется регулярное выражение `([0-9]{{3}})-(\p{{Nd}}{{3}})-([0-9]{{4}})`. Оно создает три группы, каждая из которых заключается в круглые скобки. Код `\p{{Nd}}` использует именованную группу символов `Nd` для обозначения десятичных цифр. Список именованных групп приводится в приложении Е.

Применяя таблицу стилей к списку телефонных номеров:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- phonelist.xml -->
<phonelist>
  <phonenumber>919-555-1212</phonenumber>
  <phonenumber>(919) 555-1212</phonenumber>
  <phonenumber>212.555.1212</phonenumber>
  <phonenumber>617-555-1212</phonenumber>
  <phonenumber>+86 555-1212</phonenumber>
</phonelist>
```

мы получаем следующий результат:

```
+1 (919) 555-1212
   Unrecognized phone number: (919) 555-1212
   Unrecognized phone number: 212.555.1212
+1 (617) 555-1212
   Unrecognized phone number: +86 555-1212
```

Если регулярное выражение совпадает со строкой, `regex-group(1)` представляет код зоны, `regex-group(2)` — код телефонной станции, а `regex-group(3)` — последние четыре цифры телефонного номера.

Если вам требуется получить всю строку, с которой совпало регулярное выражение, используйте запись `regex-group(0)`. (Также можно воспользоваться обозначением `".".`) В предыдущем примере `regex-group(0)` для первого телефонного номера возвращает `919-555-1212`.

В некоторых местах группы цифр в телефонных номерах принято разделять точками вместо дефисов. Чтобы преобразовывались и эти номера, можно изменить регулярное выражение так, чтобы оно искало совпадения и для точек, и для дефисов:

```
<xsl:analyze-string select=".'
  regex="([0-9]{{3}})(-|\.)([0-9]{{3}})(-|\.)([0-9]{{4}})">
```

Дефисы в исходном выражении заменены альтернативой `(-|.)`. Следующее выражение совпадает не только с первым и четвертым, но и с третьим элементом `<phonenumber>`, однако результат все равно получается неверным:

```
+1 (919) --555
   Unrecognized phone number: (919) 555-1212
+1 (212) .-555
+1 (617) --555
   Unrecognized phone number: +86 555-1212
```

Проблема в том, что в регулярном выражении создаются еще две группы. Необходимо изменить таблицу стилей так, чтобы в ней использовались `regex-group(1)`, `regex-group(3)` и `regex-group(5)`:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- analyze-string2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:for-each select="phonelist/phonenumber">
      <xsl:analyze-string select="."
        regex="([0-9]{{3}})(-|\.)([0-9]{{3}})(-|\.)([0-9]{{4}})">
        <xsl:matching-substring>
          <xsl:text>&#xA;+1 (</xsl:text>
          <xsl:value-of select="regex-group(1)"/>
          <xsl:text>) </xsl:text>
          <xsl:value-of select="regex-group(3)"/>
          <xsl:text>-</xsl:text>
          <xsl:value-of select="regex-group(5)"/>
        </xsl:matching-substring>
        <xsl:non-matching-substring>
          <xsl:text>&#xA;   Unrecognized phone number: </xsl:text>
          <xsl:value-of select="."/>
        </xsl:non-matching-substring>
      </xsl:analyze-string>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Теперь обновленная таблица стилей дает правильный результат:

```
+1 (919) 555-1212
   Unrecognized phone number: (919) 555-1212
+1 (212) 555-1212
+1 (617) 555-1212
   Unrecognized phone number: +86 555-1212
```

В описании флага `i` упоминался пример со знаком шкалы Кельвина (`K`) как специального символа Юникода, совпадающего с буквой `K`. Следующая таблица стилей позволяет убедиться в этом совпадении:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- analyze-string3.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:analyze-string select="'&#x212A;'"
      regex="k" flags="i">
      <xsl:matching-substring>
        <xsl:text>It matches!</xsl:text>
      </xsl:matching-substring>
      <xsl:non-matching-substring>
        <xsl:text>It doesn't match!</xsl:text>
      </xsl:non-matching-substring>
    </xsl:analyze-string>
  </xsl:template>

</xsl:stylesheet>
```

Последний пример показывает, что происходит с неверно записанным регулярным выражением. Ранее уже упоминалось о том, что все фигурные скобки (`{` и `}`) должны удваиваться. Регулярное выражение `[0-9]{{5}}` совпадает с числом из пяти цифр, а неверно записанное выражение `[0-9]{5}` — с числом из двух цифр, завершающимся цифрой `5`. (Шаблон значения атрибута `{5}` интерпретируется как строка `5`.) Таблица стилей выглядит так:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- analyze-string4.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Test of an incorrectly written regex:&#xA;&#xA;</xsl:text>
    <xsl:call-template name="bad-regex">
      <xsl:with-param name="test-value" select="'37174'"/>
    </xsl:call-template>
    <xsl:call-template name="bad-regex">
      <xsl:with-param name="test-value" select="'95'"/>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="bad-regex">
    <xsl:param name="test-value" required="yes"/>
```

```
    <xsl:value-of select="$test-value"/>
    <xsl:analyze-string select="$test-value" regex="[0-9]{5}">
      <xsl:matching-substring>
        <xsl:text> matches!&#xA;</xsl:text>
      </xsl:matching-substring>
      <xsl:non-matching-substring>
        <xsl:text> doesn't match!&#xA;</xsl:text>
      </xsl:non-matching-substring>
    </xsl:analyze-string>
  </xsl:template>

</xsl:stylesheet>
```

Применение этой таблицы стилей к любому документу XML дает следующий результат:

```
Test of an incorrectly written regex:

37174 doesn't match!
95 matches!
```

Неверно записанное выражение совпадает со значением `95`, а с предполагавшимся `37174` не совпадает.

См. также

В приложении E подробно описана работа регулярных выражений в XPath 2.0. Также см. определения следующих элементов и функций: *[2.0]* `matches()`, *[2.0]* `<xsl:matching-substring>`, *[2.0]* `<xsl:non-matching-substring>`, *[2.0]* `regex-group()`, *[2.0]* `replace()` и *[2.0]* `tokenize()`.

<xsl:apply-imports>

Позволяет применить к текущему узлу шаблон, импортированный из другой таблицы стилей.

Категория

Инструкция.

Обязательные атрибуты

Нет.

Необязательные атрибуты

Нет.

Содержимое

[1.0] Нет. Элемент `<xsl:apply-imports>` является пустым элементом.

[2.0] В XSLT 2.0 элемент `<xsl:apply-imports>` может содержать нуль или более элементов `<xsl:with-param>` для передачи параметров импортированному шаблону. Лишние параметры, передаваемые импортируемому шаблону, игнорируются. Но если импортированному шаблону не передается обязательный параметр, процессор XSLT выдает ошибку.

Местонахождение

Элемент <xsl:apply-imports> находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 5.6 «Overriding Template Rules».

[2.0] XSLT, раздел 6.7 «Overriding Template Rules»,

Пример

Короткий файл XML для демонстрации применения элемента <xsl:apply-imports>:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- codelisting.xml -->
<chapter>
  <title>Some really great code</title>
  <para>Here is one of my favorite code listings:</para>
  <programlisting>
public class HelloWorld
{
  public static void main(String[] args)
  {
    System.out.println("Hello, World!");
  }
}
  </programlisting>
  <para>I wrote that code all by myself!</para>
</chapter>
```

Главная таблица стилей содержит шаблоны для четырех элементов DocBook (<chapter>, <title>, <para> и <programlisting>) из документа XML:

```
<?xml version="1.0"?>
<!-- apply-imports.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:import href="imported.xsl"/>

  <xsl:preserve-space elements="programlisting"/>

  <xsl:output method="html"/>

  <xsl:template match="chapter">
    <html>
      <head>
        <title>
          <xsl:value-of select="title"/>
        </title>
      </head>
      <body>
        <xsl:apply-templates select="*"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="title">
    <h1>
      <xsl:value-of select="."/>
```

```
    </h1>
  </xsl:template>

  <xsl:template match="para">
    <p>
      <xsl:apply-templates select="*|text()"/>
    </p>
  </xsl:template>

  <xsl:template match="programlisting">
    <div style="font-size: 125%; font-weight:bold;
                font-family: monospace;">
      <xsl:apply-imports />
    </div>
  </xsl:template>

</xsl:stylesheet>
```

Импортируемая таблица стилей:

```
<?xml version="1.0"?>
<!-- imported.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="programlisting">
    <pre>
      <xsl:value-of select="."/>
    </pre>
  </xsl:template>

</xsl:stylesheet>
```

Импортируемая таблица стилей заключает все найденные элементы `<programlisting>` **в теги** `<pre>`**. Главная таблица стилей помещает** `<programlisting>` **в теги** `<div>`**, изменяющие шрифтовые свойства. При обработке документа XML с использованием главной таблицы стилей будет получен следующий результат:**

```
<!-- codelisting.html -->
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Some really great code</title>
   </head>
   <body>
      <h1>Some really great code</h1>
      <p>Here is one of my favorite code listings:</p>
      <div style="font-size: 125%; font-weight:bold;
                  font-family: monospace;"><pre>
public class HelloWorld
{
  public static void main(String[] args)
  {
    System.out.println("Hello, World!");
  }
}
  </pre></div>
      <p>I wrote that all by myself!</p>
   </body>
</html>
```

Вывод HTML содержит как элемент , созданный главной таблицей стилей, так и элемент <pre>, созданный импортированной таблицей (рис. А.1). Если заменить элемент <xsl:apply-imports /> элементом <xsl:value-of select="."/>, листинг «сольется» в одну строку.

***Рис. А.1.** Код HTML, сгенерированный с использованием импортированного шаблона*

См. также

Описание элемента *[2.0]* <xsl:next-match>.

<xsl:apply-templates>

Приказывает процессору XSLT применить соответствующие шаблоны к набору узлов или последовательности.

Категория

Инструкция.

Обязательные атрибуты

Нет.

Необязательные атрибуты

select

Содержит выражение XPath для выбора узлов, к которым должны быть применены шаблоны. В число допустимых значений входит обозначение * для выбора всех дочерних *элементов* текущего узла. Без этого атрибута <xsl:apply-templates> выбирает все дочерние узлы текущего узла, включая текст, инструкции по обработке и комментарии. Инструкции <xsl:apply-templates /> и <xsl:apply-templates select="node()"/> эквивалентны.

mode

Определяет *режим обработки*; этот удобный синтаксис позволяет писать специализированные шаблоны для конкретных целей. Например, можно написать элемент <xsl:template> с атрибутом mode="toc", чтобы обработать узел для включения в сводку содержимого документа, а затем написать другие элементы <xsl:template> с атрибутами mode="print", mode="online", mode="index" и т. д. для обрабатки той же информации в других целях.

[2.0] В XSLT 2.0 у атрибута mode при использовании с элементом <xsl:apply-templates> появились два специальных значения:

#default – соответствует режиму по умолчанию;

#current – соответствует текущему режиму.

Содержимое

Элемент <xsl:apply-templates> может содержать произвольное количество элементов <xsl:sort> и <xsl:with-param>. Достаточно часто элемент <xsl:apply-templates> является пустым.

Местонахождение

Элемент <xsl:apply-templates> находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 5.4 «Applying Template Rules».

[2.0] XSLT, раздел 6.3 «Applying Template Rules».

Пример

Следующая таблица стилей обрабатывает одни и те же узлы тремя разными способами. Каждый шаблон активизируется с атрибутом mode. В качестве исходного документа будет использоваться список машин:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- cars.xml -->
<cars>
  <manufacturer name="Chevrolet">
    <car>Cavalier</car>
    <car>Corvette</car>
    <car>Impala</car>
    <car>Malibu</car>
  </manufacturer>
  <manufacturer name="Ford">
    <car>Pinto</car>
    <car>Mustang</car>
    <car>Taurus</car>
  </manufacturer>
  <manufacturer name="Volkswagen">
    <car>Beetle</car>
    <car>Jetta</car>
    <car>Passat</car>
    <car>Touraeg</car>
  </manufacturer>
</cars>
```

Таблица стилей:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- apply-templates1.xsl -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Using the mode attribute</title>
      </head>
      <body style="font-family: sans-serif;">
        <table style="text-align: center;" border="1">
          <tr style="font-weight: bold; font-size: 150%;">
            <td width="30%">Default mode</td>
            <td width="30%">Blue mode</td>
            <td width="30%">Red mode</td>
          </tr>
          <tr>
            <td>
              <p>
                <xsl:apply-templates
                  select="/cars/manufacturer"/>
              </p>
            </td>
            <td>
              <p>
                <xsl:apply-templates mode="blue"
                  select="/cars/manufacturer"/>
              </p>
            </td>
            <td>
              <p>
                <xsl:apply-templates mode="red"
                  select="/cars/manufacturer"/>
              </p>
            </td>
          </tr>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="manufacturer">
    <div style="color: green; font-style: italic; font-size: 125%;">
      <xsl:apply-templates select="car"/>
    </div>
  </xsl:template>

  <xsl:template match="manufacturer" mode="blue">
    <div style="color: blue; font-weight: bold;">
      <xsl:apply-templates select="car"/>
    </div>
  </xsl:template>
```

```
  <xsl:template match="manufacturer" mode="red">
    <div style="color: red; font-family: monospace; font-weight: bold;
                font-size: 150%;">
      <xsl:apply-templates select="car"/>
    </div>
  </xsl:template>

  <xsl:template match="car">
    <xsl:value-of select="."/>
    <br/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей генерирует документ HTML, показанный на рис. А.2.

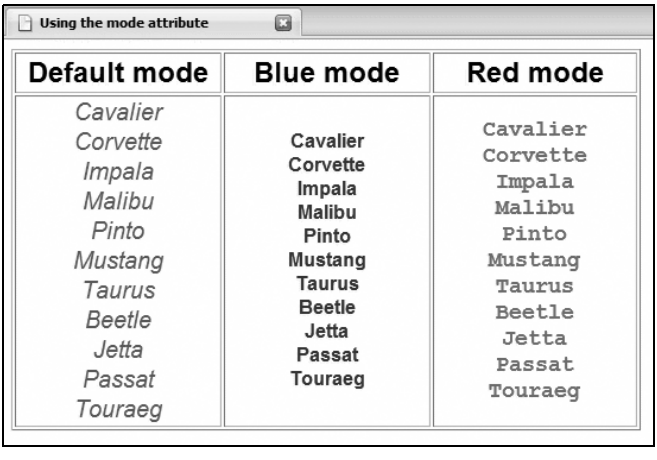


***Рис. А.2.** Код HTML, сгенерированный шаблонами с разными значениями mode*

Первый шаблон не имеет атрибута `mode`; он является шаблоном по умолчанию.

[2.0] В XSLT 2.0 возможности атрибута `mode` были расширены. В следующей таблице стилей используются значения `#default`, `#current` и `#all` атрибута `mode`:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- apply-templates2.xsl -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Using the mode attribute</title>
      </head>
      <body style="font-family: sans-serif;">
        <table style="text-align: center;" border="1">
          <tr style="font-weight: bold; font-size: 150%;">
```

```
            <td width="30%">Default mode</td>
            <td width="30%">Blue mode</td>
            <td width="30%">Red mode</td>
          </tr>
          <tr>
            <td>
              <p>
                <xsl:apply-templates mode="#default"
                  select="/cars/manufacturer"/>
              </p>
            </td>
            <td>
              <p>
                <xsl:apply-templates mode="blue"
                  select="/cars/manufacturer"/>
              </p>
            </td>
            <td>
              <p>
                <xsl:apply-templates mode="red"
                  select="/cars/manufacturer"/>
              </p>
            </td>
          </tr>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="manufacturer">
    <div style="color: green; font-style: italic; font-size: 125%">
      <xsl:apply-templates select="car" mode="#current"/>
    </div>
  </xsl:template>

  <xsl:template match="manufacturer" mode="blue red">
    <div style="color: blue; font-weight: bold;">
      <xsl:apply-templates select="car" mode="#current"/>
    </div>
  </xsl:template>

  <xsl:template match="car" mode="#all">
    <xsl:value-of select="."/>
    <br/>
  </xsl:template>

  <xsl:template match="car" mode="red" priority="1">
    <div style="color: red; font-size: 125%; font-family: serif;">
      <xsl:value-of select="."/>
      <br/>
    </div>
  </xsl:template>

</xsl:stylesheet>
```

В XSLT 2.0 атрибут `mode` **элемента** `<xsl:apply-templates>` **может принимать новые значения** `#current` **и** `#default`**. У атрибута** `mode` **элемента** `<xsl:template>` **определены новые значения** `#all` **и** `#default`**. В нашей таблице стилей используются все три новых значения:**

- В шаблоне `match="/"` элемент `<xsl:apply-templates>` используется со значениями `#default`, `blue` и `red`. Три элемента `<xsl:apply-template>` отличаются только одним – значением атрибута `mode`.
- Первый шаблон `match="manufacturer"` не имеет атрибута `mode`, поэтому он применяется в режиме по умолчанию. При использовании `<xsl:apply-templates mode="#default" ...>` будет активизирован именно этот шаблон.
- У второго шаблона `match="manufacturer"` атрибут `mode` равен `blue red`. Этот шаблон активизируется при выборе режима `blue` или `red`.
- Оба шаблона `match="manufacturer"` используют запись `<xsl:apply-templates mode="#current" ...>` для обработки элементов `<car>`. Фактически текущее значение `mode` передается процессору XSLT в виде параметра, сообщая ему, какой шаблон следует применять.
- Первый шаблон `match="car"` определяется с атрибутом `mode="#all"`, поэтому он является шаблоном по умолчанию, который активизируется обоими элементами `<xsl:apply-templates mode="#current" ...>`.
- Последний шаблон `match="car"` определяется с атрибутом `mode="red"`. Чтобы ему отдавалось предпочтение перед шаблоном `mode="#all"`, мы добавляем атрибут `priority="1"`. Без него процессор Saxon выводит предупреждение о неоднозначном применении правил, хотя и активизирует шаблон `mode="red"` при действующем режиме `red`. Ядро Altova XML никаких предупреждений не выдает – оно просто активизирует шаблон с `mode="red"`.

Начиная с шаблона `match="/"`, наша таблица стилей генерирует выходной документ вызовами разных шаблонов с тремя режимами `mode`. Три режима форматируют одну и ту же информацию тремя разными способами. В итоге таблица стилей создает тот же документ HTML, что и предыдущая таблица стилей.

См. также

Описание элемента `<xsl:template>`.

<xsl:attribute>

Создает атрибут в выходном документе. Инструкция `<xsl:attribute>` позволяет определить имя или содержимое атрибута во время выполнения. Имя или значение атрибута можно строить из частей входного документа, жестко закодированного текста, значений, возвращаемых функциями, глобальных переменных и любых других значений, к которым вы можете обратиться из своей таблицы стилей. Можно также воспользоваться элементом `<xsl:if>` для принятия решения о том, должен ли вообще создаваться атрибут в данном случае.

Категория

Инструкция.

Обязательные атрибуты

`name`

Атрибут `name` определяет имя атрибута, создаваемого элементом `<xsl:attribute>`. (Как ни старайся, а описание атрибутов элемента `<xsl:attribute>` выглядит запутанно, не правда ли?)

Необязательные атрибуты

`namespace`

Атрибут `namespace` определяет URI-адрес пространства имен, которое должно использоваться для данного атрибута в выходном документе. Вы не можете управлять префиксом пространства имен; атрибут `namespace` позволяет задать только URI-адрес.

[2.0] `select`

Выражение XPath, определяющее содержимое атрибута. Если элемент `<xsl:attribute>` содержит атрибут `select`, то сам элемент должен быть пустым.

[2.0] `separator`

Определяет символы для разделения множественных значений, генерируемых инструкцией `<xsl:attribute>`. Если элемент `<xsl:attribute>` содержит атрибут `select`, по умолчанию в качестве разделителя используется пробел (`#x20`). Без атрибута `select` по умолчанию используется строка нулевой длины (`""`). Атрибут `separator` переопределяет значение по умолчанию независимо от того, присутствует атрибут `select` или нет.

[2.0 – Schema] `type`

Определяет тип данных атрибута. Им может быть любой из встроенных типов данных или же тип данных, определенный в схеме (при использовании схемосовместимого процессора XSLT 2.0).

Атрибуты `type` и `validation` являются взаимоисключающими.

[2.0 – Schema] `validation`

Определяет способ проверки значения нового атрибута. Атрибут `validation` имеет четыре допустимых значения: `strict`, `lax`, `preserve` и `strip`. Поскольку мы создаем атрибут, а не элемент или итоговый документ, некоторые стандартные особенности атрибута `validation` здесь не действуют.

Значения `strip` и `preserve` в данном случае ни на что не влияют. Они соответственно указывают, должно ли обозначение типа атрибута удаляться или сохраняться в сгенерированном атрибуте. Для атрибутов проверка не выполняется, и они имеют обозначение типа `xs:untypedAtomic`.

Атрибут `validation="strict"` означает, что процессор XSLT должен искать во всех объявленных схемах глобальное объявление атрибута (`<xs:attribute>`) с таким же именем, как у данного атрибута. Если процессору не удается найти подходящий элемент `<xs:attribute>`, происходит фатальная ошибка. Если же процессор находит объявление атрибута, он проверяет по нему сгенерированное значение.

Последнее значение `validation="lax"` работает аналогично `validation="strict"`, но если процессору не удается найти объявление атрибута ни в одной из объявленных схем, ошибка не происходит. В этом случае атрибут имеет обозначение типа `xs:untypedAtomic`.

Атрибуты `validation` и `type` являются взаимоисключающими.

Содержимое

Содержимое атрибута строится из любых элементов, генерирующих *[1.0]* набор узлов или *[2.0]* последовательность. Элементы `<xsl:choose>`, `<xsl:text>`, `<xsl:value-of>` и *[2.0]* `<xsl:sequence>` удовлетворяют этому условию.

Местонахождение

Элемент <xsl:attribute> находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 7.1.3 «Creating Attributes with xsl:attribute».

[2.0] XSLT, раздел 11.3 «Creating Attribute Nodes Using xsl:attribute».

Пример

В следующем примере мы построим таблицу HTML по данным из следующего документа XML:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Мы хотим создать таблицу, состоящую из двух столбцов: в правом столбце содержимое каждого элемента <listitem> выводится в отдельной ячейке, а левый столбец состоит из одной ячейки с атрибутом rowspan, равным количеству элементов <listitem> в документе XML. Естественно, значение атрибута rowspan не может жестко кодироваться, поскольку количество элементов <listitem> является переменной величиной. Задача решается при помощи элемента <xsl:attribute>:

```
<?xml version="1.0"?>
<!-- attribute1.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of select="list/title"/></title>
      </head>
      <body style="font-family: sans-serif;">
        <xsl:apply-templates select="list"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="list">
    <table border="1" cellpadding="5" cellspacing="5">
      <tr>
        <td style="background: black; color: white;
                   font-weight: bold; font-size: 125%;"
                   width="100" align="right">
```

```
          <xsl:if test="count(listitem) > 1">
            <xsl:attribute name="rowspan">
              <xsl:value-of select="count(listitem)"/>
            </xsl:attribute>
          </xsl:if>
          <xsl:value-of select="title"/>
        </td>
        <td>
          <xsl:value-of select="listitem[1]"/>
        </td>
      </tr>
      <xsl:for-each select="listitem[position() > 1]">
        <tr>
          <td>
            <xsl:value-of select="."/>
          </td>
        </tr>
      </xsl:for-each>
    </table>
  </xsl:template>

</xsl:stylesheet>
```

Сгенерированный документ HTML:

```
<!-- attribute.html -->
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Albums I've bought recently:</title>
   </head>
   <body style="font-family: sans-serif;">
      <table border="1" cellpadding="5" cellspacing="5">
         <tr>
            <td style="background: black; color: white;
                       font-weight: bold; font-size: 125%;"
                       width="100" align="right" rowspan="7">
              Albums I've bought recently:</td>
            <td>The Sacred Art of Dub</td>
         </tr>
         <tr>
            <td>Only the Poor Man Feel It</td>
         </tr>
         ...
      </table>
   </body>
</html>
```

Обратите внимание: элемент `<td>` содержит несколько жестко закодированных атрибутов; эти атрибуты объединяются с атрибутом, создаваемым элементом `<xsl:attribute>`. Элементов `<xsl:attribute>` может быть сколько угодно, но все атрибуты `<xsl:attribute>` должны располагаться перед всем остальным содержимым шаблона. На рис. А.3 показано, как сгенерированный документ HTML выглядит в броузере.

Учтите, что в данном примере можно было также воспользоваться шаблоном значения атрибута. Значение атрибута `rowspan` генерируется следующим образом:

```
<td bgcolor="black" rowspan="{count(listitem)}"
  width="100" align="right">
```

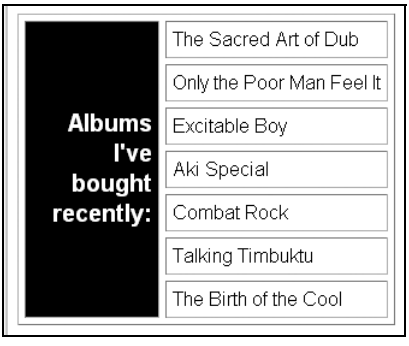


Рис. А.3. *Документ со сгенерированными атрибутами*

Выражение в фигурных скобках ({}) вычисляется и заменяется своим итоговым значением. В данном случае вызов `count(listitem)` возвращает число `7`, которое и становится значением атрибута `rowspan`. Единственное отличие здесь заключается в том, что `rowspan` генерируется и при наличии только одного элемента `<listitem>`.

[2.0] Следующий короткий пример демонстрирует применение новых атрибутов `select` и `separator`. В таблице стилей диапазонное выражение XPath 2.0 (выражение `"to"`) используется для присваивания последовательности значений атрибуту `example`:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- attribute2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" omit-xml-declaration="yes"/>

  <xsl:template match="/">
    <sampledoc>
      <xsl:attribute name="example" select="1 to 7" separator=", "/>
    </sampledoc>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей генерирует очень короткий документ:

```
<sampledoc example="1, 2, 3, 4, 5, 6, 7"/>
```

Благодаря атрибуту XSLT 2.0 `separator` вам не придется самостоятельно реализовывать логику следующего вида:

```
<xsl:for-each>
  <xsl:value-of select="."/>
  <xsl:if test="position() != last()">
    <xsl:text>, </xsl:text>
  </xsl:if>
</xsl:for-each>
```

[2.0 – Schema] В последнем примере используются схемосовместимые атрибуты `<xsl:attribute>`. Наша таблица стилей содержит импортированную схему, опреде-

ляющую новый тип данных; атрибут <xsl:attribute> используется для создания атрибутов с этим типом данных. Таблица стилей выглядит так:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- attribute3.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:zip="http://www.oreilly.com/xslt/zip"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  exclude-result-prefixes="xs">

  <xsl:output method="xml" indent="yes"/>

  <xsl:import-schema namespace="http://www.oreilly.com/xslt/zip">
    <xsd:schema
      xmlns="http://www.oreilly.com/xslt/zip"
      targetNamespace="http://www.oreilly.com/xslt/zip"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <xsd:simpleType name="zipcode">
        <xsd:restriction base="xsd:string">
          <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:schema>
  </xsl:import-schema>

  <xsl:template match="/">
    <postcodes>
      <xsl:for-each select="postcodes/postcode">
        <postcode>
          <xsl:choose>
            <xsl:when test=". castable as zip:zipcode">
              <xsl:attribute name="zip:zip" type="zip:zipcode">
                <xsl:value-of select=". cast as zip:zipcode"/>
              </xsl:attribute>
            </xsl:when>
            <xsl:otherwise>
              <xsl:attribute name="other" type="xs:string">
                <xsl:value-of select="."/>
              </xsl:attribute>
            </xsl:otherwise>
          </xsl:choose>
        </postcode>
      </xsl:for-each>
    </postcodes>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей используется для обработки следующего документа:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- postcodes.xml -->
<postcodes>
  <postcode>3S8 EOX</postcode>
  <postcode>37174</postcode>
  <postcode>NSW 3829</postcode>
  <postcode>27516</postcode>
</postcodes>
```

Таблица стилей использует оператор XPath `castable as` для определения того, может ли значение из исходного документа быть преобразовано в значение `zip:zipcode`. Если проверка дает положительный результат, мы создаем новый атрибут с типом данных `zip:zipcode`. Результат:

```
<?xml version="1.0" encoding="UTF-8"?>
<postcodes xmlns:zip="http://www.oreilly.com/xslt/zip">
   <postcode other="3S8 EOX"/>
   <postcode zip:zip="37174"/>
   <postcode other="NSW 3829"/>
   <postcode zip:zip="27516"/>
</postcodes>
```

Два атрибута из сгенерированного набора уточняются пространством имен и содержат значения соответствующего типа данных.

<xsl:attribute-set>

Определяет группу атрибутов в выходных данных. В дальнейшем вы можете ссылаться на весь набор атрибутов по имени (вместо ссылок на отдельные атрибуты).

Категория

Элемент верхнего уровня.

Обязательные атрибуты

`name`

Определяет имя набора атрибутов.

Необязательные атрибуты

`use-attribute-sets`

Перечисляет один или несколько наборов атрибутов, которые должны использоваться текущим набором. Если вы задаете более одного набора, разделите их имена символами-пропусками. Атрибут может использоваться для включения других элементов `<xsl:attribute-set>` в текущий набор, но учтите, что прямое или косвенное включение элементом `<xsl:attribute-set>` самого себя приводит к ошибке. Иначе говоря, если набор атрибутов A включает набор атрибутов B, набор B включает набор C, а набор C включает набор A, процессор XSLT сообщает об ошибке.

Содержимое

Один или несколько элементов `<xsl:attribute>`.

Местонахождение

`<xsl:attribute-set>` относится к элементам верхнего уровня, а его родителем может быть только элемент `<xsl:stylesheet>`.

Где определяется

[1.0] XSLT, раздел 7.1.4 «Named Attribute Sets».

[2.0] XSLT, раздел 10.2 «Named Attribute Sets».

Пример

Таблица стилей будет использоваться с тремя элементами `<xsl:attribute-set>`**. В первой группе определяется атрибут** `style` **с несколькими свойствами CSS; во второй определяется пара атрибутов HTML** `<table>` **и используется первая группа; наконец, третья группа определяет традиционный элемент HTML и атрибут CSS** `style`**.**

```
<?xml version="1.0"?>
<!-- attribute-set.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:attribute-set name="bold-table">
    <xsl:attribute name="style">
      font-weight: bold;
    </xsl:attribute>
  </xsl:attribute-set>

  <xsl:attribute-set name="spacious-table"
    use-attribute-sets="bold-table">
    <xsl:attribute name="cellpadding">8</xsl:attribute>
    <xsl:attribute name="cellspacing">8</xsl:attribute>
  </xsl:attribute-set>

  <xsl:attribute-set name="reverse-table">
    <xsl:attribute name="bgcolor">black</xsl:attribute>
    <xsl:attribute name="style">color: white;</xsl:attribute>
  </xsl:attribute-set>

  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of select="/list/title"/></title>
      </head>
      <body style="font-family: sans-serif;">
        <xsl:apply-templates select="*"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="list">
    <h1><xsl:value-of select="title"/></h1>
    <table xsl:use-attribute-sets="spacious-table" border="2">
      <xsl:for-each select="listitem">
        <tr>
          <td xsl:use-attribute-sets="reverse-table">
            <xsl:value-of select="."/>
          </td>
        </tr>
      </xsl:for-each>
    </table>
    <h1>Here's the same table with different attribute sets:</h1>
    <table border="2" xsl:use-attribute-sets="bold-table">
      <xsl:for-each select="listitem">
        <tr>
          <td>
```

```
            <xsl:value-of select="."/>
          </td>
        </tr>
      </xsl:for-each>
    </table>
  </xsl:template>

</xsl:stylesheet>
```

Обратите внимание на необходимость уточнения пространства имен `xsl:use-attribute-sets` для его обработки процессором XSLT. Если просто использовать запись `use-attribute-sets`, процессор XSLT копирует атрибут в выходной документ HTML без обработки.

Наша таблица стилей создает две таблицы по списку объектов с использованием разных именованных наборов атрибутов. Таблица стилей применяется к одному из наших типовых документов XML:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list>
  <title>A few of my favorite albums</title>
  <listitem>A Love Supreme</listitem>
  <listitem>Beat Crazy</listitem>
  <listitem>You Could Have it So Much Better</listitem>
  <listitem>Kind of Blue</listitem>
  <listitem>London Calling</listitem>
  <listitem>Remain in Light</listitem>
  <listitem>The Joshua Tree</listitem>
  <listitem>The Indestructible Beat of Soweto</listitem>
</list>
```

Таблица стилей генерирует следующий код HTML:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Albums I've bought recently:</title>
   </head>
   <body>
      <h1>Albums I've bought recently:</h1>
      <table style="font-weight: bold; font-family: sans-serif;"
                  cellpadding="8" cellspacing="8" border="2">
         <tr>
            <td bgcolor="black" style="color: white;">The Sacred Art of Dub</td>
         </tr>
         ...
         <tr>
            <td bgcolor="black" style="color: white;">The Birth of the Cool</td>
         </tr>
      </table>
      <h1>Here's the same table with different attribute sets:</h1>
      <table style="font-weight: bold; font-family: sans-serif;" border="2">
         <tr>
            <td>The Sacred Art of Dub</td>
         </tr>
         ...
```

```
            <tr>
               <td>The Birth of the Cool</td>
            </tr>
         </table>
      </body>
   </html>
```

На рис. А.4 показано, как выглядят 2 сгенерированные таблицы.

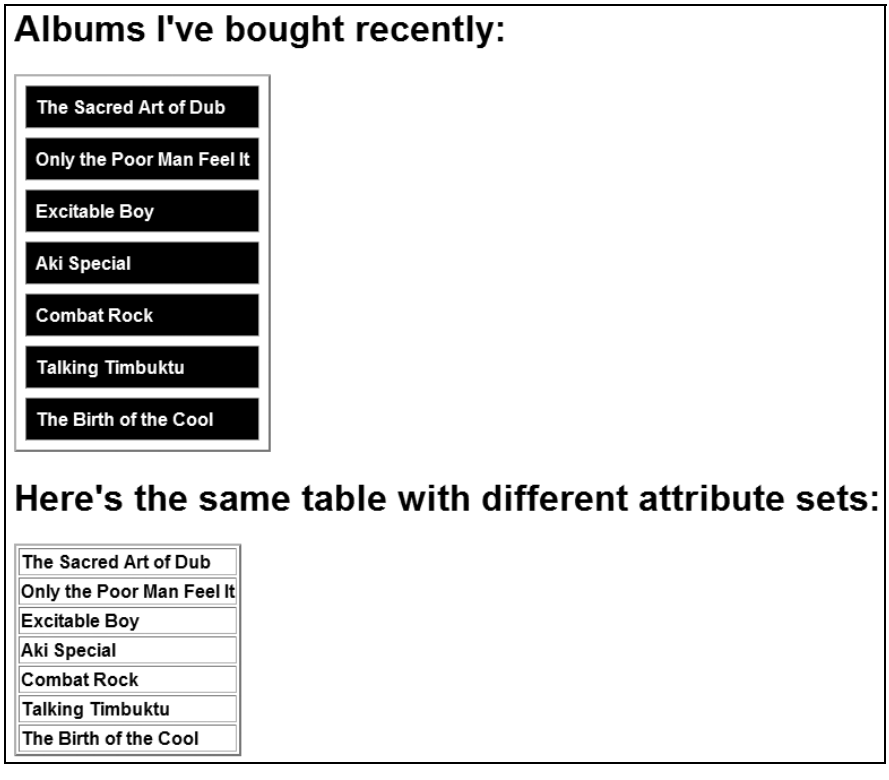


***Рис. А.4.** Таблицы, созданные с двумя наборами атрибутов*

Учтите, что все атрибуты, закодированные в элементе, обладают более высоким приоритетом, чем атрибуты, добавленные в элемент с использованием `xsl:use-attribute-sets`. Допустим, в документе присутствует следующий элемент:

```
<table cellpadding="3" xsl:use-attribute-sets="spacious-table" ...>
```

Атрибут `cellpadding` из набора атрибутов `spacious-table` не применяется. Процессор XSLT генерирует следующий элемент `<table>`:

```
<table cellpadding="3" cellspacing="8" ...>
```

<xsl:call-template>

Активизирует шаблон по имени. Если какие-то фрагменты выходных данных должны генерироваться относительно часто, вы можете выделить разметку, создающую этот вывод, в именованный шаблон и активизировать этот шаблон по мере надобности.

Категория

Инструкция.

Обязательные атрибуты

`name`

Имя активизируемого шаблона.

Необязательные атрибуты

Нет.

Содержимое

Элемент может содержать любое количество необязательных элементов `<xsl:with-param>`.

Местонахождение

Элемент `<xsl:call-template>` находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 6 «Named Templates».

[2.0] XSLT, раздел 10.1 «Named Templates».

Пример

Активизация именованных шаблонов элементом `<xsl:call-template>` – отличный способ создания модульных таблиц стилей. В следующем примере именованные шаблоны используются для форматирования документа с данными о продажах шоколада:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
```

```
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Таблица стилей с именованными шаблонами:

```
<?xml version="1.0"?>
<!-- call-template.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="report">
    <html>
      <head>
        <xsl:call-template name="report-title">
          <xsl:with-param name="in-heading" select="true()"/>
          <xsl:with-param name="title" select="title"/>
        </xsl:call-template>
      </head>
      <body>
        <xsl:call-template name="insert-header"/>
        <xsl:call-template name="report-title">
          <xsl:with-param name="in-heading" select="false()"/>
          <xsl:with-param name="title" select="title"/>
        </xsl:call-template>
        <table cellpadding="5">
          <xsl:call-template name="table-heading"/>
          <xsl:apply-templates select="brand"/>
        </table>
        <xsl:call-template name="insert-footer"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template name="insert-header">
    <p style="font-size: 75%; font-style: italic;">
      This confidential report is the property of DougCo, Inc.
    </p>
    <hr/>
  </xsl:template>

  <xsl:template name="report-title">
    <xsl:param name="in-heading"/>
    <xsl:param name="title"/>
    <xsl:choose>
      <xsl:when test="$in-heading">
        <title><xsl:value-of select="$title"/></title>
      </xsl:when>
      <xsl:otherwise>
        <h1><xsl:value-of select="$title"/></h1>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

```
  <xsl:template name="table-heading">
    <tr>
      <td style="background: black; color: white; font-weight: bold;">
        Brand</td>
      <td style="background: black; color: white; font-weight: bold;">
        Sales</td>
    </tr>
  </xsl:template>

  <xsl:template name="insert-footer">
    <hr/>
    <p style="font-size: 75%; font-style: italic;">
      © Copyright 2008, DougCo, Inc.
    </p>
  </xsl:template>

  <xsl:template match="brand">
    <tr>
      <td><xsl:value-of select="name"/></td>
      <td><xsl:value-of select="units"/></td>
    </tr>
  </xsl:template>

</xsl:stylesheet>
```

В таблице стилей используются четыре разных именованных шаблона. Шаблоны `insert-header`, `table-heading` и `insert-footer` позволяют легко вставлять часто используемую разметку в выходной документ. На практике такие шаблоны обычно импортируются из другой таблицы стилей. Если десятки отчетов, веб-страниц и других документов используют один и тот же заголовок и завершитель, сохраните эту разметку в одном месте и включайте ее в нужных местах простой инструкцией `<xsl:call-template>`.

Еще один именованный шаблон, `insert-title`, использует два параметра. Первый параметр сообщает именованному шаблону, производится ли вызов из секции `<head>` документа HTML; второй параметр содержит название отчета. В зависимости от значения `in-heading` шаблон генерирует элемент `<title>` или элемент `<h1>`.

Таблица стилей генерирует следующий файл HTML:

```
<html>
  <head>
    <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Chocolate bar sales</title>
  </head>
  <body>
    <p style="font-size: 75%; font-style: italic;">
      This confidential report is the property of DougCo, Inc.
    </p>
    <hr>
    <h1>Chocolate bar sales</h1>
    <table cellpadding="5">
      <tr>
        <td style="background: black; color: white; font-weight: bold;">
          Brand
        </td>
        <td style="background: black; color: white; font-weight: bold;">
          Sales
```

```
        </td>
      </tr>
      <tr>
        <td>Lindt</td><td>27408</td>
      </tr>
      <tr>
        <td>Callebaut</td><td>8203</td>
      </tr>
      <tr>
        <td>Valrhona</td><td>22101</td>
      </tr>
      <tr>
        <td>Perugina</td><td>14336</td>
      </tr>
      <tr>
        <td>Ghirardelli</td><td>19268</td>
      </tr>
    </table>
    <hr>
    <p style="font-size: 75%; font-style: italic;">
      &copy; Copyright 2007, DougCo, Inc.
    </p>
  </body>
</html>
```

Результат изображен на рис. А.5.

Рис. А.5. *Документ HTML, сгенерированный с использованием именованных шаблонов*

[2.0] В XSLT 2.0 правила передачи параметров шаблонам слегка изменились:

- В XSLT 1.0 шаблону можно было передать любое количество параметров; лишние параметры (т. е. параметры, не определенные в вызываемом шаблоне) игнорировались. В XSLT 2.0 это приводит к фатальной ошибке.

- В XSLT 2.0 элемент <xsl:param> поддерживает атрибут required. Вызов шаблона без передачи значений всех обязательных параметров приводит к фатальной ошибке.
- В XSLT 2.0 появилась концепция туннельных параметров. За дополнительной информацией о туннельных параметрах обращайтесь к разделу «Туннельные параметры» главы 5.

[2.0] <xsl:character-map>

Определяет карту символов – набор символов, каждый из которых должен заменяться некоторой строкой. Карты символов позволяют включать нестандартные символы в значения элементов и атрибутов. Элемент <xsl:character-map> имеет много общего с объявлением XML <!ENTITY>. Используйте <xsl:character-map> вместо атрибутов disable-output-escaping элементов <xsl:text> и <xsl:value-of>, определенных в XSLT 1.0. (Атрибут disable-output-escaping считается устаревшим в XSLT 2.0.)

Категория

Объявление.

Обязательные атрибуты

name

Имя символьной карты.

Необязательные атрибуты

use-character-maps

Разделенные пробелами имена одной или нескольких карт символов, включаемых в текущую карту. Как и следовало ожидать, включение (прямое или косвенное) картой символов самой себя приводит к фатальной ошибке. Кроме того, попытка включения несуществующей карты символов тоже приводит к фатальной ошибке.

Содержимое

Нуль или более элементов <xsl:output-character>.

Местонахождение

Элемент <xsl:character-map> является дочерним элементом <xsl:stylesheet>.

Где определяется

XSLT, раздел 20 «Serialization».

Пример

Далее определяется простой пример с парой полезных функций. Сначала мы создадим отображение, заменяющее символы табуляции (	) двумя пробелами; символы табуляции часто выводятся в виде серий из восьми пробелов, что может создать проблемы при выводе листингов с отступами. Затем мы определим пару графических изображений, которые должны выводиться вместо некоторых символов. Исходный код XML, который будет использоваться в этом примере:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- special-characters.xml -->
<char-test>
  <tabs>public class HelloWorld {
        public static void main(String[] args) {
                System.out.println("Hello, World!");
        }
}</tabs>
  <special-char>&#x2780;</special-char>
  <special-char>&#x2781;</special-char>
</char-test>
```

Таблица стилей заменяет символы табуляции двумя пробелами, а два специальных символа – графикой (диапазон символов Юникода от `➀` **до** `➉` **состоит из заключенных в круг чисел от 1 до 10, выводимых шрифтом без засечек, т. е. от ① до ⑩). Таблица стилей выглядит так:**

```
<?xml version="1.0" encoding="utf-8"?>
<!-- character-map1.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" use-character-maps="sample"/>

  <xsl:character-map name="sample" use-character-maps="circles">
    <xsl:output-character character="&#x9;" string="  "/>
  </xsl:character-map>

  <xsl:character-map name="circles">
    <xsl:output-character character="&#x2780;"
      string="&lt;img src='images/circle1.gif'
      width='28' height='28'/&gt;"/>
    <xsl:output-character character="&#x2781;"
      string="&lt;img src='images/circle2.gif'
      width='28' height='28'/&gt;"/>
  </xsl:character-map>

  <xsl:template match="char-test">
    <html>
      <head>
        <title>A test of some special characters</title>
      </head>
      <body style="font-family: sans-serif; ">
        <h1>A test of some special characters</h1>
        <xsl:apply-templates select="*"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="tabs">
    <pre style="font-sixe: 150%; font-weight: bold;">
      <xsl:value-of select="."/>
    </pre>
  </xsl:template>

  <xsl:template match="special-char">
    <p style="font-size: 200%;">
      <xsl:text>Here's a special character: </xsl:text>
      <xsl:value-of select="."/>
```

```
      </p>
    </xsl:template>

  </xsl:stylesheet>
```

Обратите внимание: таблица стилей просто выводит значения элементов `<tabs>` и `<special-char>`. Элемент `<xsl:output>` и его атрибут `use-character-maps="sample"` берут преобразование символов на себя. Элемент `<xsl:character-map>` с именем `sample` определяет одиночную замену, а также использует элемент `<xsl:character-map> name="circles"`. Поставленная задача выполняется благодаря сочетанию определения карты символов и обращения ее к элементу `<xsl:output>`.

Заметьте также, что подстановки в карте символов `circles` заменяют символ разметкой HTML – элементом `<img>`. Документ HTML выглядит так:

```
<html>
  <head>
    <meta http-eqiuv="Content-Type" content="text/html; charset=UTF-8">
    <title>A test of some special characters</title>
  </head>
  <body style="font-family: sans-serif;">
    <h1>A test of some special characters</h1>
    <pre style="font-size: 150%"
    Font-weight: bold;>public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }
}</pre><p style="font-size: 200%;">
      Here's a special character:
      <img src='images/circle1.gif' width='28' height='28'/></p>
    <p style="font-size: 200%;">
      Here's a special character:
      <img src='images/circle2.gif' width='28' height='28'/></p>
  </body>
</html>
```

На рис. А.6 показано, как эти символы выглядят в броузере.

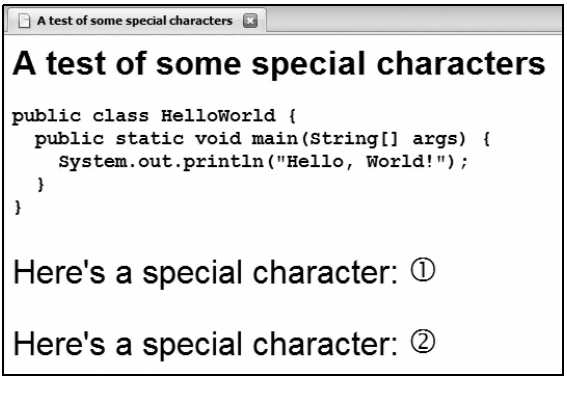


Рис. А.6. *Код HTML, сгенерированный с использованием карт символов*

Элемент <xsl:character-map> часто применяется также для преобразования символа новой строки (
) в пару символов возврата курсора и перевода строки (
), но этот пример не будет наглядным. Если вам потребуется сделать подобное, элемент <xsl:output-character> должен выглядеть примерно так:

```
<xsl:output-character character="&#xA;" string="&#xD;&#xA;"/>
```

Наконец, в спецификации XSLT 2.0 приводится пример с созданием JSP (Java Server Pages). Использование угловых скобок в JSP нарушает совместимость с XML и HTML. Мы можем определить карту символов, которая выглядит примерно так:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- character-map2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" use-character-maps="jsp"/>

  <xsl:character-map name="jsp">
    <xsl:output-character character="«" string="&lt;%="/>
    <xsl:output-character character="»" string="%&gt;"/>
  </xsl:character-map>

  <xsl:template match="jsp-test">
    <html>
      <head>
        <title>A test of some special characters</title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>Generating a .jsp page</h1>
        <p>Here's the value of a JSP function:
          <xsl:apply-templates select="text()"/>
        </p>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Для входного файла XML:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- jsp-test.xml -->
<jsp-test>« new java.util.Date() »</jsp-test>
```

генерируется следующий результат:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>A test of some special characters</title>
   </head>
   <body style="font-family: sans-serif;">
      <h1>Generating a .jsp page</h1>
      <p>Here's the value of a JSP function:
         <%= new java.util.Date() %>
      </p>
   </body>
</html>
```

<xsl:choose>

Элемент <xsl:choose> реализует логику if-then-else в XSLT.

Категория

Инструкция.

Обязательные атрибуты

Нет.

Необязательные атрибуты

Нет.

Содержимое

Содержит один или несколько элементов <xsl:when>. Также может содержать один элемент <xsl:otherwise>. Если этот элемент присутствует, он должен быть последним элементом в <xsl:choose>.

Местонахождение

Элемент <xsl:choose> находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 9.2 «Conditional Processing with xsl:choose».

[2.0] XSLT, раздел 8.2 «Conditional Processing with xsl:choose».

Пример

В следующем примере элемент <xsl:choose> применяется для выбора цвета фона строк таблицы HTML. Таблица стилей перебирает четыре разных значения, а <xsl:choose> используется для определения значения атрибута style в сгенерированном документе HTML. Документ XML выглядит так:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Таблица стилей:

```
<?xml version="1.0"?>
<!-- choose.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>
```

```
  <xsl:template match="/">
    <html>
      <head>
        <title>
          <xsl:value-of select="list/title"/>
        </title>
      </head>
      <body style="font-family: sans-serif; color: white;">
        <h1 style="color: black;">
          <xsl:value-of select="list/title"/>
        </h1>
        <table border="1" cellpadding="5"
          style="font-weight: bold;">
          <xsl:for-each select="list/listitem">
            <tr>
              <td>
                <xsl:attribute name="style">
                  <xsl:choose>
                    <xsl:when test="position() mod 4 = 0">
                      <xsl:text>background: yellow; color: black;</xsl:text>
                    </xsl:when>
                    <xsl:when test="position() mod 4 = 1">
                      <xsl:text>background: blue;</xsl:text>
                    </xsl:when>
                    <xsl:when test="position() mod 4 = 2">
                      <xsl:text>background: white; color: black;</xsl:text>
                    </xsl:when>
                    <xsl:otherwise>
                      <xsl:text>background: black;</xsl:text>
                    </xsl:otherwise>
                  </xsl:choose>
                </xsl:attribute>
                <xsl:value-of select="."/>
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Элемент `<xsl:choose>` **используется при генерировании атрибута** `style` **каждого сгенерированного элемента** `<td>`**. Пример сгенерированного документа HTML с перебором цветов фона:**

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Albums I've bought recently:</title>
   </head>
   <body style="font-family: sans-serif; color: white;">
      <h1 style="color: black;">Albums I've bought recently:</h1>
      <table border="1" cellpadding="5" style="font-weight: bold;">
         <tr>
            <td style="background: blue;">The Sacred Art of Dub</td>
         </tr>
```

```
            ...
            <tr>
               <td style="background: black;">The Birth of the Cool</td>
            </tr>
         </table>
      </body>
   </html>
```

На рис. А.7 показано, как сгенерированный документ HTML выглядит в броузере.

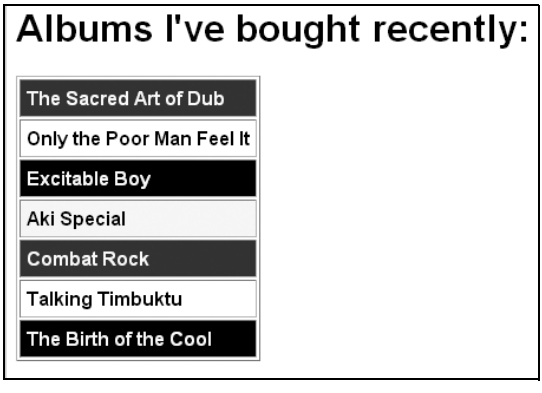


Рис. А.7. *Использование элемента <xsl:choose> для перебора цветов фона*

<xsl:comment>

Включает комментарии в выходной документ. В комментариях обычно приводятся правовая информация, юридические предупреждения, дата создания выходного документа и т. д. Элемент `<xsl:comment>` может также использоваться для построения определений CSS или кода JavaScript в документе HTML.

Категория

Инструкция.

Обязательные атрибуты

Нет.

Необязательные атрибуты

[2.0] `select`

Выражение XPath, генерирующее содержимое комментария. Если атрибут отсутствует, вместо него используется содержимое элемента `<xsl:comment>`. Если атрибут `select` отсутствует, а элемент `<xsl:comment>` пуст, генерируется пустой комментарий. Если элемент содержит атрибут `select` и одновременно имеет содержимое, происходит фатальная ошибка.

Содержимое

Конструктор *[1.0]* набора узлов или *[2.0]* последовательности.

Местонахождение

Элемент <xsl:comment> находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 7.4 «Creating Comments».

[2.0] XSLT, раздел 11.8 «Creating Comments».

Пример

Следующая таблица стилей генерирует комментарий для определения стилей CSS в документе HTML:

```
<?xml version="1.0"?>
<!-- comment.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>XSLT and CSS Demo</title>
        <style>
          <xsl:comment>
            p.big   {font-size: 125%; font-weight: bold;}
            p.odd   {color: purple; font-weight: bold;}
            p.even  {color: blue; font-style: italic; font-weight: bold;}
          </xsl:comment>
        </style>
      </head>
      <body style="font-family: sans-serif;">
        <xsl:apply-templates select="list/title"/>
        <xsl:apply-templates select="list/listitem"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="title">
    <p class="big"><xsl:value-of select="."/></p>
  </xsl:template>

  <xsl:template match="listitem">
    <xsl:choose>
      <xsl:when test="position() mod 2">
        <p class="odd"><xsl:value-of select="."/></p>
      </xsl:when>
      <xsl:otherwise>
        <p class="even"><xsl:value-of select="."/></p>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

</xsl:stylesheet>
```

Эта таблица стилей создает три стиля CSS в комментарии HTML. Она применяется к следующему документу:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Таблица стилей применяет один стиль CSS к элементу `<title>`, а затем последовательно чередует два других стиля CSS для элементов `<listitem>`. Сгенерированный код HTML выглядит так:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>XSLT and CSS Demo</title><style>
         <!--
            p.big      {font-size: 125%; font-weight: bold;}
            p.odd      {color: purple; font-weight: bold;}
            p.even     {color: blue; font-style: italic; font-weight: bold;}
          --></style></head>
   <body style="font-family: sans-serif;">
      <p class="big">Albums I've bought recently:</p>
      <p class="odd">The Sacred Art of Dub</p>
      <p class="even">Only the Poor Man Feel It</p>
      <p class="odd">Excitable Boy</p>
      <p class="even">Aki Special</p>
      <p class="odd">Combat Rock</p>
      <p class="even">Talking Timbuktu</p>
      <p class="odd">The Birth of the Cool</p>
   </body>
</html>
```

На рис. А.8 показано, как сгенерированный документ HTML выглядит в броузере.

Рис. А.8. *Документ со свойствами CSS, определенными в узлах комментария*

[2.0] Чтобы использовать атрибут `select`, добавленный в XSLT 2.0, можно включить элемент `<xsl:comment>` в таблицу стилей:

```
<xsl:comment select="concat('The second album is ', list/listitem[2])"/>
```

В выходных данных HTML генерируется следующий комментарий:

```
<!--The second album is Only the Poor Man Feel It-->
```

Если элемент `<xsl:comment>` используется для генерирования кода JavaScript, завершите свой код комментарием JavaScript, как в следующем примере:

```
<SCRIPT type="text/javascript">
<xsl:comment> // Скрыть код от старых броузеров
  function iOver(image)
  {
    if (browser="N3") document[image].src=eval(image + "over.src");
  }
  // </xsl:comment>
</SCRIPT>
```

Размещение двух косых черт в последней строке создает комментарий JavaScript. В этом случае интерпретатор JavaScript проигнорирует знак `-->` в конце комментария.

<xsl:copy>

По умолчанию элемент `<xsl:copy>` создает поверхностную копию узла. Копируются только текущий узел и узлы его пространства имен; узлы атрибутов и дочерние узлы не копируются. Элемент `<xsl:copy>` дает возможность точной настройки процесса копирования, но это потребует дополнительных усилий. Элемент `<xsl:copy>` может использоваться для копирования любых узлов, включая узлы комментариев и атрибутов.

Категория

Инструкция.

Обязательные атрибуты

Нет.

Необязательные атрибуты

`use-attribute-sets`

Перечисляет один или несколько наборов атрибутов, которые должны использоваться элементом. Если вы задаете более одного набора, разделите их имена символами-пропусками. За дополнительной информацией обращайтесь к описанию элемента `<xsl:attribute-set>`.

[2.0] `copy-namespaces`

Определяет, должны ли копироваться пространства имен. Применяется только при копировании узлов элементов. Допустимые значения: `yes` (используется по умолчанию) и `no`.

[2.0] inherit-namespaces

Определяет, должны ли элемент и его дочерние элементы наследовать текущие узлы пространств имен. Допустимые значения: yes (используется по умолчанию) и no.

[2.0 – Schema] type

Определяет тип данных копируемого элемента. Им может быть любой из встроенных типов данных или же тип данных, определенный в схеме (при использовании схемосовместимого процессора XSLT 2.0).

Атрибуты type и validation являются взаимоисключающими.

[2.0 – Schema] validation

Определяет способ проверки значения нового атрибута. Атрибут validation имеет четыре допустимых значения: strict, lax, preserve и strip.

Атрибут validation="strict" означает, что процессор XSLT должен искать во всех объявленных схемах глобальное объявление атрибута или элемента (<xs:attribute> или <xs:element>) с таким же именем, как у копируемого узла. Если процессору не удается найти подходящее объявление, происходит фатальная ошибка. Если же процессор находит объявление узла, он проверяет по нему значение копируемого узла.

Значение validation="lax" работает аналогично validation="strict", но если процессору не удается найти объявление копируемого узла ни в одной из объявленных схем, ошибка не происходит. В этом случае элемент имеет обозначение типа xs:untyped.

Эффект значения validation="preserve" зависит от типа копируемого узла. Если копируемый узел является атрибутом, его обозначение типа сохраняется. Для узлов элементов скопированный узел имеет обозначение типа xs:anyType. Обозначения типов всех узлов, содержащихся в копируемом элементе, сохраняются. Проверка по схеме не производится.

Значение validation="strip" заменяет обозначения типа копируемых узлов атрибутов и элементов на xs:untypedAtomic и xs:untyped соответственно. У всех узлов атрибутов и элементов, содержащихся в копируемом узле, обозначения типов заменяются на xs:untypedAtomic и xs:untyped соответственно.

Атрибуты validation и type являются взаимоисключающими.

Содержимое

Шаблон XSLT.

Местонахождение

Элемент <xsl:copy> находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 7.5 «Copying».

[2.0] XSLT, раздел 11.9.1 «Shallow Copying».

Пример

Для демонстрации применения <xsl:copy> рассмотрим копирование элемента в дерево результата. Первая таблица стилей просто копирует элемент документа:

```
<?xml version="1.0"?>
<!-- copy1.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml"/>

  <xsl:template match="*">
    <xsl:copy/>
  </xsl:template>

</xsl:stylesheet>
```

Для тестирования таблицы стилей будет использоваться следующий документ XML:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Так как мы ничего не сделали для копирования дочерних элементов и атрибутов элемента документа, результат получается очень коротким:

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

Чтобы получить более содержательный результат, необходимо скопировать дочерние элементы каждого элемента. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- copy2.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml"/>

  <xsl:template match="*">
    <xsl:copy>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
<?xml version="1.0" encoding="UTF-8"?><list>
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem>Aki Special</listitem>
  <listitem>Combat Rock</listitem>
  <listitem>Talking Timbuktu</listitem>
  <listitem>The Birth of the Cool</listitem>
</list>
```

Результат содержит больше данных, но в полученном документе нет атрибутов. Элемент <xsl:copy> выполняет поверхностное копирование, но позволяет полностью управлять выводом (в отличие от <xsl:copy-of>). Оборотной стороной такого управления является то, что вам придется явно указывать все копируемые дочерние узлы или узлы атрибутов. (Поскольку узлы атрибутов не считаются дочерними узлами своих родительских элементов, их придется выбирать и копировать отдельно.) Таблица стилей не копирует также узлы комментариев и инструкций по обработке; для них придется включать в таблицу дополнительную разметку.

Сравните этот пример с примером для элемента <xsl:copy-of>.

Инструкция <xsl:copy> может оказаться полезной также в шаблоне, совпадающем с несколькими элементами. Возьмем документ с данными о продажах шоколада:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Таблица стилей использует <xsl:copy> для того, чтобы из документа копировались только элементы <brand> и <name>:

```
<?xml version="1.0"?>
<!-- copy3.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml"/>

  <xsl:template match="report">
    <brands>
      <xsl:apply-templates select="brand"/>
    </brands>
  </xsl:template>

  <xsl:template match="brand|name|units">
    <xsl:copy>
```

```
        <xsl:apply-templates/>
      </xsl:copy>
    </xsl:template>

</xsl:stylesheet>
```

Таблица стилей создает новый элемент `<brands>`, а затем копирует все элементы `<brand>` с их дочерними элементами. Результат:

```
<?xml version="1.0" encoding="UTF-8"?><report><brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand><brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand><brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand><brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand><brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand></report>
```

Обратите внимание: текстовые узлы выбранных элементов обрабатываются правилом таблиц стилей по умолчанию, копирующим текст в выходной документ.

<xsl:copy-of>

Создает глубокую копию узла.

Категория

Инструкция.

Обязательные атрибуты

`select`

Содержит выражение XPath, определяющее узлы, копируемые в выходной документ.

Необязательные атрибуты

[2.0] `copy-namespaces`

Определяет, должны ли копироваться пространства имен. Применяется только при копировании узлов элементов. Допустимые значения: `yes` (используется по умолчанию) и `no`.

[2.0 – Schema] `type`

Определяет тип данных копируемого узла. Им может быть любой из встроенных типов данных или же тип данных, определенный в схеме (при использовании схемосовместимого процессора XSLT 2.0).

Атрибуты `type` и `validation` являются взаимоисключающими.

[2.0 – Schema] validation

Определяет способ проверки значения нового атрибута. Атрибут validation имеет четыре допустимых значения: strict, lax, preserve и strip.

Атрибут validation="strict" означает, что процессор XSLT должен искать во всех объявленных схемах объявление узла (<xs:attribute> или <xs:element>) с таким же именем, как у копируемого узла. Если процессору не удается найти подходящее объявление, происходит фатальная ошибка. Если же процессор находит объявление узла, он проверяет по нему значение копируемого узла.

Значение validation="lax" работает аналогично validation="strict", но если процессору не удается найти объявление копируемого узла ни в одной из объявленных схем, ошибка не происходит. В этом случае скопированному атрибуту или элементу присваивается обозначение типа xs:untypedAtomic или xs:untyped соответственно.

Со значением validation="preserve" все скопированные узлы сохраняют свои обозначения типов.

Наконец, значение validation="strip" удаляет все обозначения типов из всех скопированных узлов. Скопированные узлы атрибутов будут иметь обозначение типа xs:untypedAtomic, а скопированные узлы элементов – обозначение типа xs:untyped.

Атрибуты validation и type являются взаимоисключающими.

Содержимое

Нет. Элемент <xsl:copy-of> является пустым элементом.

Местонахождение

Элемент <xsl:copy-of> находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 11.3 «Using Values of Variables and Parameters with xsl:copy-of».

[2.0] XSLT, раздел 11.9.2 «Deep Copy».

Пример

Для демонстрации применения элемента <xsl:copy-of> используется простая таблица стилей, копирующая входной документ в дерево результата:

```
<?xml version="1.0"?>
<!-- copy-of.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml"/>

  <xsl:template match="/">
    <xsl:copy-of select="."/>
  </xsl:template>

</xsl:stylesheet>
```

Тестовый документ:

```
<?xml version="1.0"?>
<!-- albums.xml -->
```

```
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Результат преобразования почти неотличим от исходного документа XML:

```
<?xml version="1.0" encoding="UTF-8"?><!-- albums.xml --><list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Различия между двумя документами минимальны: из результата удалена часть пропусков, а ядро XSLT включило в объявление XML информацию о кодировке. На семантическом уровне эти два документа идентичны. Сравните с аналогичным примером для элемента `<xsl:copy>`.

Учтите, что в XSLT 1.0 и 2.0 поведение этого элемента различно:

[1.0]:

- Если атрибут `select` определяет фрагмент дерева результата, то весь фрагмент копируется в дерево результата.
- Если атрибут `select` определяет набор узлов, то все узлы набора копируются в дерево результата в порядке документа. В отличие от `<xsl:copy>`, узел копируется полностью – со всеми узлами пространств имен, атрибутов и дочерними узлами.
- Если атрибут `select` определяет нечто отличное от фрагмента дерева результата или набора узлов, это «нечто» преобразуется в строку и вставляется в дерево результата.

[2.0]:

- Если атрибут `select` определяет элемент, то этот элемент со всеми потомками и атрибутами копируется в выходной документ. По умолчанию узлы пространства имен элемента тоже копируются, хотя это поведение можно изменить при помощи атрибута `copy-namespaces`.
- Если атрибут `select` указывает на узел документа, этот узел документа со всеми потомками копируется в выходной документ.
- Все остальные типы узлов (узлы атрибутов, пространств имен, текста, комментариев и инструкций по обработке) копируются в выходной документ.
- Атомарные значения присоединяются к итоговой последовательности.

<xsl:decimal-format>

Определяет числовой формат, который должен использоваться при записи числовых значений в выходной документ. Если элемент <decimal-format> не содержит атрибута name, предполагается числовой формат по умолчанию, используемый при всех вызовах функции format-number(). С другой стороны, если числовому формату присвоено имя (name), оно может передаваться при вызове функции format-number().

Категория

Элемент верхнего уровня.

Обязательные атрибуты

Нет.

Необязательные атрибуты

name

Определяет имя формата.

decimal-separator

Определяет символ (обычно точка или запятая), используемый для отделения дробной части (как в форматной строке, так и в выводе). По умолчанию используется символ «точка» (.).

grouping-separator

Определяет символ (обычно точка или запятая), используемый в качестве разделителя групп разрядов (как в форматной строке, так и в выводе). По умолчанию используется символ «запятая» (,).

infinity

Определяет строку, используемую для представления бесконечности (только в выводе). Учтите, что математическое ядро XSLT поддерживает как положительную, так и отрицательную бесконечность. По умолчанию используется строка "Infinity".

minus-sign

Определяет символ, используемый в качестве знака «минус» (только в выводе). По умолчанию используется знак «дефис» (-, -).

NaN

Определяет строку, отображаемую в том случае, если форматируемое значение не является числом (только в выводе). По умолчанию используется строка "NaN".

percent

Определяет символ для обозначения процентов (как в форматной строке, так и в выводе). По умолчанию используется знак процента (%).

per-mille

Определяет символ для обозначения промилле (как в форматной строке, так и в выводе). По умолчанию используется символ промилле в Юникоде (‰, ‰).

zero-digit

Определяет символ для обозначения цифры «нуль» (как в форматной строке, так и в выводе). По умолчанию используется цифра «нуль» (0).

`digit`

Определяет символ для обозначения цифры в форматной строке. По умолчанию используется знак «решетка» (#).

`pattern-separator`

Определяет символ для разделения подшаблонов положительного и отрицательного представления числа (только в форматной строке). По умолчанию используется символ «точка с запятой» (;).

Содержимое

Нет. Элемент <xsl:decimal-format> является пустым элементом.

Местонахождение

<xsl:decimal-format> относится к элементам верхнего уровня, а его родителем может быть только элемент <xsl:stylesheet>.

Где определяется

[1.0] XSLT, раздел 12.3 «Number Formatting».

[2.0] XSLT, раздел 16.4.1 «Defining a Decimal Format».

Пример

Таблица стилей с определениями двух элементов <decimal-format>:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- decimal-format.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

f  <xsl:decimal-format name="f1"
    decimal-separator=":"
    grouping-separator="/"/>

  <xsl:decimal-format name="f2"
    infinity="Really, really big"
    NaN="[not a number]"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the &lt;decimal-format&gt; element:</xsl:text>

    <xsl:text>&#xA;&#xA;    format-number(1528.3, '#/###:00', 'f1')=</xsl:text>
    <xsl:value-of select="format-number(1528.3, '#/###:00;-#/###:00', 'f1')"/>
    <xsl:text>&#xA;    format-number(1 div 0, '###,###.00', 'f2')=</xsl:text>
    <xsl:value-of select="format-number(1 div 0, '###,###.00', 'f2')"/>
    <xsl:text>&#xA;    format-number(blue div orange, '#.##', 'f2')=</xsl:text>
    <xsl:value-of select="format-number(blue div orange, '#.##', 'f2')"/>
    <xsl:text>&#xA;&#xA;**********************************</xsl:text>
    <xsl:text>&#xA;Sales report for </xsl:text>
    <xsl:value-of select="/report/@month"/>
    <xsl:text>/</xsl:text>
    <xsl:value-of select="/report/@year"/>
    <xsl:text>&#xA;&#xA;</xsl:text>
    <xsl:variable name="totalSales" select="sum(/report/brand/units)"/>
    <xsl:for-each select="report/brand">
```

```
      <xsl:value-of select="name"/>
      <xsl:text>: &#xA;    </xsl:text>
      <xsl:value-of select="format-number(units, '##,###')"/>
      <xsl:text> bars sold, </xsl:text>
      <xsl:value-of select="format-number(units div $totalSales, '##%')"/>
      <xsl:text> of all sales.</xsl:text>
      <xsl:text>&#xA;</xsl:text>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
    <xsl:text>Total sales: </xsl:text>
    <xsl:value-of select="format-number($totalSales, '##,###')"/>
    <xsl:text> bars.&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей применяется к следующему документу:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

При обработке документа и таблицы стилей процессором XSLT 1.0 будет получен следующий результат:

```
Tests of the <decimal-format> element:

    format-number(1528.3, '#/###:00', 'f1')=1/528:30
    format-number(1 div 0, '###,###.00', 'f2')=Really, really big
    format-number(blue div orange, '#.##', 'f2')=[not a number]

**************************************
Sales report for 8/2006

Lindt:
    27,408 bars sold, 30% of all sales.
```

```
Callebaut:
    8,203 bars sold, 9% of all sales.

Valrhona:
    22,101 bars sold, 24% of all sales.

Perugina:
    14,336 bars sold, 16% of all sales.

Ghirardelli:
    19,268 bars sold, 21% of all sales.

Total sales: 91,316 bars.
```

[2.0] Как неоднократно упоминалось ранее, в XSLT 2.0 устанавливаются более жесткие ограничения синтаксиса таблиц стилей. Например, таблица стилей с командой `1 div 0` работать не будет. Помните, что в любой элемент XSLT 2.0 можно включить атрибут `version`, если требуется использовать поведение XSLT 1.0 в таблице стилей XSLT 2.0:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- decimal-format2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
...
    <xsl:text>&#xA;    format-number(1 div 0, '###,###.00', 'f2')=</xsl:text>
    <xsl:value-of version="1.0"
      select="format-number(1 div 0, '###,###.00', 'f2')"/>
...
</xsl:stylesheet>
```

Элементу `<xsl:stylesheet>` задается атрибут `version` со значением `2.0`, однако позднее в элемент `<xsl:value-of>` включается атрибут `version="1.0"`. В результате поведение XSLT 1.0 активизируется только для одного элемента, а это означает, что при делении на нуль будет получено значение `Really, really big`. При использовании стандартного поведения XSLT 2.0 таблица стилей выдает исключение и осмысленный вывод не генерируется.

В XSLT 2.0 деление `xs:double` на нуль возвращает строку `Infinity`, а деление `xs:integer` или `xs:decimal` на нуль приводит к ошибке времени выполнения. Если привести выражение в таблице стилей к виду `xs:double(1) div xs:double(0)`, таблица стилей отработает без ошибок и выдаст тот же результат (`Infinity`), что и таблица стилей XSLT 1.0 с выражением `1 div 0`.

За дополнительной информацией об атрибуте `version` и других атрибутах, которые могут присутствовать в любом элементе XSLT, обращайтесь к разделу «[2.0] Атрибуты, общие для всех элементов XSLT» в начале этого приложения.

[2.0] <xsl:document>

Создает новый узел документа. Элемент может использоваться, например, для проверки узла документа по схеме. Узел документа, созданный элементом `<xsl:document>`, не предназначен для сериализации (записи на диск); для создания сериализуемого узла документа следует использовать элемент `<xsl:result-document>`.

Категория

Инструкция.

Обязательные атрибуты

Нет.

Необязательные атрибуты

[2.0 – Schema] `type`

Определят тип данных корневого узла элемента, создаваемого этим элементом. Чтобы узел документа прошел проверку на действительность, его единственным дочерним элементом должен быть один узел элемента, без текстовых узлов и с нулем или более узлами комментариев и инструкций по обработке. Если структура узла документа не соответствует этому описанию, процессор XSLT выдает ошибку.

Атрибуты `type` и `validation` являются взаимоисключающими.

[2.0 – Schema] `validation`

Определяет способ проверки значения нового атрибута. Атрибут `validation` имеет четыре допустимых значения: `strict`, `lax`, `preserve` и `strip`. Чтобы узел документа прошел проверку на действительность в режимах `strict` и `lax`, его единственным дочерним элементом должен быть один узел элемента, без текстовых узлов и с нулем или более узлами комментариев и инструкций по обработке. Если в режиме `strict` или `lax` структура узла документа не соответствует этому описанию, процессор XSLT выдает ошибку.

Атрибут `validation="strict"` означает, что процессор XSLT должен искать во всех объявленных схемах глобальное объявление элемента (`<xs:element>`) с таким же именем, как у единственного узла элемента, являющегося дочерним узлом этого узла документа. Если процессору не удается найти подходящее объявление `<xs:element>`, происходит фатальная ошибка. Если же процессор находит объявление узла элемента, он проверяет по нему сгенерированное значение.

Значение `validation="lax"` работает аналогично `validation="strict"`, но если процессору не удается найти объявление элемента ни в одной из объявленных схем, ошибка не происходит. В этом случае элемент имеет обозначение типа `xs:untyped`.

Со значением `validation="preserve"` обозначения типов единственного узла элемента и всех его дочерних элементов и атрибутов сохраняются без изменений. Проверка по схеме не выполняется.

Значение `validation="strip"` назначает единственному узлу элемента обозначение типа `xs:untyped`. Всем дочерним элементам и атрибутам задаются обозначения типов `xs:untyped` и `xs:untypedAtomic` соответственно. Проверка по схеме не выполняется.

Атрибуты `validation` и `type` являются взаимоисключающими.

Содержимое

Конструктор последовательности.

Местонахождение

Элемент `<xsl:document>` находится в шаблоне.

Где определяется

[2.0] XSLT, раздел 11.5 «Creating Document Nodes».

Пример

В следующем примере мы создадим узел документа и зададим его содержимое. Согласно определению из схемы документа с заказами этим узлом будет элемент <name>:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- po.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="purchase-order">
    ...
  </xs:element>
  ...

  <xs:element name="name">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="title"
          minOccurs="0" maxOccurs="1"/>
        <xs:element ref="first-name"
          minOccurs="1" maxOccurs="1"/>
        <xs:element ref="last-name"
          minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="title" type="xs:string"/>
  <xs:element name="first-name" type="xs:string"/>
  <xs:element name="last-name" type="xs:string"/>

</xs:schema>
```

Таблица стилей:

```
<?xml version="1.0"?>
<!-- document.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.oreilly.com/xslt"
  xmlns:po="http://www.oreilly.com/xslt">

  <xsl:output method="xml" indent="yes"/>

  <xsl:import-schema namespace="http://www.oreilly.com/xslt"
    schema-location="po.xsd" />

  <xsl:template match="/">
    <xsl:document validation="lax">
      <xsl:element name="name">
        <xsl:element name="title">
          <xsl:text>Mr.</xsl:text>
        </xsl:element>
        <xsl:element name="first-name">
```

```
          <xsl:text>Kent Lyle</xsl:text>
        </xsl:element>
        <xsl:element name="last-name">
          <xsl:text>Birdley</xsl:text>
        </xsl:element>
      </xsl:element>
    </xsl:document>
  </xsl:template>

</xsl:stylesheet>
```

В этом примере используется режим `validation="lax"`. Это означает, что процессор XSLT подтвердит действительность узла документа, содержащего элемент `<name>`, если ему удастся найти схему XML с определением элемента `<name>`. Если убрать элемент `<xsl:import-schema>`, элемент `<xsl:document>` все равно будет работать. Но если убрать элемент `<xsl:import-schema>` и перевести `<xsl:document>` в режим `validation="strict"`, попытка выполнения таблицы стилей завершится неудачей, потому что процессор не находит объявление `<name>`.

Результаты применения таблицы стилей:

```
<?xml version="1.0" encoding="UTF-8"?>
<name xmlns="http://www.oreilly.com/xslt">
   <title>Mr.</title>
   <first-name>Kent Lyle</first-name>
   <last-name>Birdley</last-name>
</name>
```

В данном случае содержимое элемента `<xsl:document>` просто записывается в выходной поток. В более стандартном варианте использования `<xsl:document>` узел документа сохраняется в переменной, а затем эта переменная, прошедшая проверку действительности, используется в другом месте таблицы стилей.

Элемент `<xsl:document>` может использоваться для создания узла документа, который не является корректным (well-formed). Пример соответствующей таблицы стилей:

```
<?xml version="1.0"?>
<!-- document2.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:variable name="ill-formed" as="node()*">
    <xsl:document>
      <xsl:element name="title">
        <xsl:text>Mr.</xsl:text>
      </xsl:element>
      <xsl:element name="first-name">
        <xsl:text>Kent Lyle</xsl:text>
      </xsl:element>
      <xsl:element name="last-name">
        <xsl:text>Birdley</xsl:text>
      </xsl:element>
    </xsl:document>
  </xsl:variable>

  <xsl:template match="/">
```

```
    <xsl:text>&#xA;A document node that isn't well formed:</xsl:text>
    <xsl:text>&#xA;&#xA;  Is this a document node? </xsl:text>
    <xsl:value-of select="if ($ill-formed instance of document-node())
                          then 'Yes!'
                          else 'No!'"/>
    <xsl:text>&#xA;  Number of child elements: </xsl:text>
    <xsl:value-of select="count($ill-formed/*)"/>

    <xsl:result-document method="xml" href="ill-formed.xml">
      <xsl:copy-of select="$ill-formed"/>
    </xsl:result-document>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей создает переменную с именем `$ill-formed`, содержащую три элемента в корне документа. При помощи теста узла `document-node()` мы убеждаемся в том, что узел действительно является узлом документа, а затем выводим количество его дочерних элементов:

```
A document node that isn't well formed:

  Is this a document node? Yes!
  Number of child elements: 3
```

Наконец, элемент `<xsl:result-document>` записывает недействительную разметку в файл. Файл `ill-formed.xml` выглядит так:

```
<?xml version="1.0" encoding="UTF-8"?>
<title>Mr.</title>
<first-name>Kent Lyle</first-name>
<last-name>Birdley</last-name>
```

Обычно узел документа имеет один дочерний узел, который представляет элемент документа; в данном случае недопустимая разметка создается в обход парсера XML. Хотя такая ситуация не типична, следует помнить, что таблица стилей может сгенерировать узел документа, имеющий более одного дочернего узла.

<xsl:element>

Создает элемент в выходном документе. Работает по аналогии с элементом `<xsl:attribute>`.

Категория

Инструкция.

Обязательный атрибут

`name`

Определяет имя элемента. Значение `name="Fred"` создает элемент `<Fred>` в выходном документе.

Необязательные атрибуты

`namespace`

Определяет пространство имен для создаваемого элемента.

use-attribute-sets

Перечисляет один или несколько наборов атрибутов, которые должны использоваться элементом. Если вы задаете более одного набора, разделите их имена символами-пропусками.

[2.0] inherit-namespaces

Определяет, должен ли элемент и его дочерние элементы наследовать текущие узлы пространств имен. Допустимые значения: yes (используется по умолчанию) и no.

[2.0 – Schema] type

Определяет тип данных элемента. Им может быть любой из встроенных типов данных или же тип данных, определенный в схеме (при использовании схемосовместимого процессора XSLT 2.0).

Атрибуты type и validation являются взаимоисключающими.

[2.0 – Schema] validation

Определяет способ проверки значения нового элемента. Атрибут validation имеет четыре допустимых значения: strict, lax, preserve и strip.

Атрибут validation="strict" означает, что процессор XSLT должен искать во всех объявленных схемах глобальное объявление элемента (<xs:element>) с таким же именем, как у нового элемента. Если процессору не удается найти подходящее объявление <xs:element>, происходит фатальная ошибка. Если же процессор находит объявление узла элемента, он проверяет по нему значение сгенерированного элемента.

Значение validation="lax" работает аналогично validation="strict", но если процессору не удается найти объявление элемента ни в одной из объявленных схем, ошибка не происходит. В этом случае элемент имеет обозначение типа xs:untyped.

Со значением validation="preserve" созданный элемент получает обозначение типа xs:anyType, а обозначения типов всех узлов, содержащихся в новом элементе, сохраняются без изменений. Проверка по схеме не выполняется.

Значение validation="strip" создает элемент с типом xs:anyType, а у всех узлов, содержащихся в новом элементе, обозначения типов заменяются на xs:untyped для элементов и xs:untypedAtomic для атрибутов. Проверка по схеме не выполняется.

Атрибуты validation и type являются взаимоисключающими.

Содержимое

Шаблон XSLT.

Местонахождение

Элемент <xsl:element> находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 7.1.2 «Creating Elements with xsl:element».

[2.0] XSLT, раздел 11.2 «Creating Element Nodes using xsl:element».

Пример

В следующем примере используется обобщенная таблица стилей, которая копирует входной документ в дерево результата с одним исключением: все атрибуты исходно-

го документа преобразуются в дочерние элементы в дереве результата. Имя атрибута становится именем нового элемента, а значение атрибута – его текстом. Так как имя атрибута неизвестно до обработки исходного документа XML, для создания дерева результата необходимо использовать элемент <xsl:element>. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- element.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml"/>

  <xsl:template match="*">
    <xsl:element name="{name()}">
      <xsl:for-each select="@*">
        <xsl:element name="{name()}">
          <xsl:value-of select="."/>
        </xsl:element>
      </xsl:for-each>
      <xsl:apply-templates select="*|text()"/>
    </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

В этой таблице стилей элемент <xsl:element> используется в двух местах: сначала для создания нового элемента с таким же именем, как у исходного элемента, а затем для создания нового элемента с таким же именем, как у каждого атрибута. Таблица стилей применяется к следующему документу:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Результат выглядит так:

```
<?xml version="1.0" encoding="UTF-8"?><list><xml:lang>en</xml:lang>
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem><xml:lang>sw</xml:lang>Aki Special</listitem>
  <listitem><xml:lang>en-gb</xml:lang>Combat Rock</listitem>
  <listitem><xml:lang>zu</xml:lang>Talking Timbuktu</listitem>
  <listitem><xml:lang>jz</xml:lang>The Birth of the Cool</listitem>
</list>
```

Элемент <xsl:element> создал все элементы в выходном документе, включая элементы <xml:lang>, созданные на базе атрибутов xml:lang.

[2.0 – Schema] Мы переходим к примеру, в котором используется поддержка схем в XSLT 2.0. Атрибут `validation` элемента `<xsl:element>` поможет убедиться в том, что создаваемый элемент действителен в соответствии со схемой заказа. Мы начнем с файла XML, содержимое которого по формату сходно с данными заказов:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- create-po.xml -->
<po order-num="38292">
  <customer id="4738" standing="Platinum">
    <address>
      <name>
        <courtesy>Mr.</courtesy>
        <given-name>Chester Hasbrouck</given-name>
        <surname>Frisby</surname>
      </name>
      <street>1234 Main Street</street>
      <city>Sheboygan</city>
      <state>WI</state>
      <zip>48392</zip>
    </address>
  </customer>
  <line-items>
    <line-item>
      <partnum>28392-33-TT</partnum>
      <partname>Turnip Twaddler</partname>
      <quantity>3</quantity>
      <price>9.95</price>
    </line-item>
    <line-item>
      <partnum>28100-38-CT</partnum>
      <partname>Clam Teaser</partname>
      <quantity>7</quantity>
      <price>39.95</price>
    </line-item>
  </line-items>
</po>
```

Элемент `<xsl:element>` будет использоваться для создания нового заказа на основании данных из документа. Необходимо также сгенерировать элемент `<date>`, являющийся первым дочерним элементом `<purchase-order>`. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- element2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.oreilly.com/xslt"
  xmlns:po="http://www.oreilly.com/xslt"
  exclude-result-prefixes="xs po">

  <xsl:import-schema namespace="http://www.oreilly.com/xslt"
    schema-location="po.xsd" />

  <xsl:output method="xml" indent="yes"/>

  <xsl:variable name="now" as="xs:date" select="current-date()"/>

  <xsl:template match="po">
    <xsl:element name="purchase-order" validation="strict">
```

```
    <xsl:attribute name="id" select="@order-num"/>
    <date>
      <xsl:attribute name="year" select="year-from-date($now)"/>
      <xsl:attribute name="month" select="month-from-date($now)"/>
      <xsl:attribute name="day" select="day-from-date($now)"/>
    </date>
    <customer>
      <xsl:attribute name="id" select="customer/@id"/>
      <xsl:attribute name="level" select="customer/@standing"/>
      <xsl:apply-templates select="customer/address"/>
    </customer>
    <xsl:apply-templates select="line-items"/>
  </xsl:element>
</xsl:template>

<xsl:template match="address">
  <address>
    <xsl:attribute name="type" select="'business'"/>
    <xsl:apply-templates select="name"/>
    <street>
      <xsl:value-of select="street"/>
    </street>
    <city>
      <xsl:value-of select="city"/>
    </city>
    <state>
      <xsl:value-of select="state"/>
    </state>
    <zip>
      <xsl:value-of select="zip"/>
    </zip>
  </address>
</xsl:template>

<xsl:template match="line-items">
  <items>
    <xsl:for-each select="line-item">
      <item>
        <xsl:attribute name="part-no" select="partnum"/>
        <partname>
          <xsl:value-of select="partname"/>
        </partname>
        <qty>
          <xsl:value-of select="quantity"/>
        </qty>
        <price>
          <xsl:value-of select="price"/>
        </price>
      </item>
    </xsl:for-each>
  </items>
</xsl:template>

<xsl:template match="name">
  <name>
    <xsl:if test="courtesy">
      <title>
```

```
          <xsl:value-of select="courtesy"/>
        </title>
      </xsl:if>
      <first-name>
        <xsl:value-of select="given-name"/>
      </first-name>
      <last-name>
        <xsl:value-of select="surname"/>
      </last-name>
    </name>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей использует элементы <xsl:element> и <xsl:attribute> для построения заказа «с нуля». Атрибут validation="strict" означает, что сгенерированный элемент <purchase-order> должен представлять действительный заказ при проверке по схеме *po.xsd*. Если таблица стилей попытается добавить в сгенерированный элемент <purchase-order> что-либо, нарушающее схему заказа, ядро XSLT инициирует исключение и прерывает работу. Таблица стилей генерирует следующий код XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<purchase-order xmlns="http://www.oreilly.com/xslt" id="38292">
   <date year="2008" month="3" day="2"/>
   <customer id="4738" level="Platinum">
      <address type="business">
         <name>
            <title>Mr.</title>
            <first-name>Chester Hasbrouck</first-name>
            <last-name>Frisby</last-name>
         </name>
         <street>1234 Main Street</street>
         <city>Sheboygan</city>
         <state>WI</state>
         <zip>48392</zip>
      </address>
   </customer>
   <items>
      <item part-no="28392-33-TT">
         <partname>Turnip Twaddler</partname>
         <qty>3</qty>
         <price>9.95</price>
      </item>
      <item part-no="28100-38-CT">
         <partname>Clam Teaser</partname>
         <qty>7</qty>
         <price>39.95</price>
      </item>
   </items>
</purchase-order>
```

Сгенерированный документ использует пространство имен по умолчанию http://www.oreilly.com/xslt. Это пространство имен связано с импортированной схемой; тот факт, что оно объявляется как пространство имен по умолчанию для таблицы стилей, означает, что оно будет скопировано в выходной элемент. Обратите также внимание на использование в таблице стилей атрибута exclude-result-prefixes, из-за которого префиксы po и xs не попадают в сгенерированный документ. Как и следовало

ожидать, пространством имен по умолчанию для сгенерированного заказа является пространство имен `http://www.oreilly.com/xslt`.

При создании элемента `<date>` используется функция `current-date()`. Чтобы увидеть, что произойдет в случае неудачной проверки, удалите элемент `<date>`. Появится следующее сообщение об ошибке:

```
XTTE1510: In content of element <purchase-order>: The content model does
  not allow element <customer> to appear here. Expected:
  {http://www.oreilly.com/xslt}date (See
  http://www.w3.org/TR/xmlschema-1/#cvc-complex-type clause 2.4)
Transformation failed: Run-time errors were reported
```

В этом случае элемент `<purchase-order>` не генерируется, потому что без находящегося в положенном месте элемента `<date>` он недействителен.

<xsl:fallback>

Определяет шаблон, который должен использоваться, если процессор XSLT обнаруживает в таблице стилей инструкцию, которую он не может обработать. Обработка `<xsl:fallback>` инициируется, например, элементом расширения, зависящим от кода, который не удается найти. Шаблон `<xsl:fallback>` может также активизироваться при обнаружении процессором XSLT 1.0 элемента XSLT 2.0.

Категория

Инструкция.

Обязательные атрибуты

Нет.

Необязательные атрибуты

Нет.

Содержимое

Шаблон XSLT.

Местонахождение

Элемент `<xsl:fallback>` находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 15 «Fallback».

[2.0] XSLT, раздел 18 «Extensibility and Fallback».

Пример

В следующей таблице стилей элемент `<xsl:fallback>` завершает преобразование при неудачном поиске элемента расширения:

```
<?xml version="1.0"?>
<!-- fallback.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
    xmlns:db="xalan://DatabaseExtension"
    extension-element-prefixes="db">

    <xsl:output method="html"/>

    <xsl:template match="/">
      <html>
        <head>
          <title><xsl:value-of select="report/title"/></title>
        </head>
        <body>
          <h1><xsl:value-of select="report/title"/></h1>
          <xsl:for-each select="report/section">
            <h2><xsl:value-of select="title"/></h2>
            <xsl:for-each select="dbaccess">
              <db:accessDatabase>
                <xsl:fallback>
                  <p>Sorry, the database library is not available!</p>
                </xsl:fallback>
              </db:accessDatabase>
            </xsl:for-each>
          </xsl:for-each>
        </body>
      </html>
    </xsl:template>

</xsl:stylesheet>
```

Таблица стилей применяется к следующему документу XML:

```
<?xml version="1.0"?>
<!-- use-db.xml -->
<report>
  <title>HR employee listing</title>
  <section>
    <title>Employees by department</title>
    <dbaccess driver="COM.ibm.db2.jdbc.app.DB2Driver"
      database="jdbc:db2:sample" tablename="employee" where="*"
      fieldnames='workdept as "Department", lastname as "Last Name",
                  firstnme as "First Name"'
      order-by="workdept" group-by="workdept, lastname, firstnme"/>
  </section>
</report>
```

В ходе преобразования документа по этой таблице стилей элемент `<xsl:fallback>` обрабатывается в том случае, если процессору не удается найти элемент расширения. Если элемент недоступен, вы получите следующий результат:

```
<html>
  <head>
    <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>HR employee listing</title>
  </head>
  <body>
    <h1>HR employee listing</h1>
    <h2>Employees by department</h2>
    <p>Sorry, the database library is not available!</p>
  </body>
</html>
```

В данном примере используется элемент расширения <dbaccess>. Исходная версия примера была написана для процессора Xalan; в Xalan значение URI-адреса префикса элемента расширения представляет собой имя класса Java. Если по какой-либо причине класс DatabaseExtension загрузить не удается, вместо него обрабатывается элемент <xsl:fallback>.

Учтите, что элемент <xsl:fallback> обрабатывается только в том случае, если элемент расширения не найден. Если код реализации элемента расширения найден, но в процессе его работы происходит сбой, ошибка должна быть обработана другим способом. Учтите также, что корректность завершения обработки таблицы стилей зависит от конкретного процессора XSLT.

<xsl:for-each>

Итеративный оператор XSLT. Этот элемент обладает атрибутом select, который выбирает узлы из текущего контекста. В дальнейшем содержимое элемента <xsl:for-each> обрабатывается для каждого выбранного узла. (*[2.0]* В XSLT 2.0 атрибут select выбирает объекты, к числу которых могут относиться как атомарные значения, так и узлы.)

Категория

Инструкция.

Обязательные атрибуты

select

Содержит выражение XPath, выбирающее узлы из текущего контекста.

Необязательные атрибуты

Нет.

Содержимое

Элемент <xsl:for-each> содержит шаблон, обрабатываемый для каждого выбранного узла. Элемент <xsl:for-each> может содержать один или несколько элементов <xsl:sort> для упорядочения выбранных узлов перед обработкой. Все элементы <xsl:sort> должны находиться в самом начале содержимого, перед всеми остальными элементами.

Местонахождение

Элемент <xsl:for-each> находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 8 «Repetition».

[2.0] XSLT, раздел 7 «Repetition».

Пример

Для демонстрации применения элемента <xsl:for-each> используется следующая таблица стилей:

```
<?xml version="1.0"?>
<!-- for-each.xsl -->
```

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Here is a moderately long list:
</xsl:text>
    <xsl:variable name="listitems" select="list/listitem"/>
    <xsl:call-template name="processListitems">
      <xsl:with-param name="items" select="$listitems"/>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="processListitems">
    <xsl:param name="items"/>
    <xsl:for-each select="$items">
      <xsl:value-of select="position()"/>
      <xsl:text>.  </xsl:text>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

В этой таблице стилей для демонстрации применения элемента <xsl:for-each> используется элемент <xsl:param> с именем items. Параметр items содержит некоторое количество элементов <listitem> из исходного документа XML; элемент <xsl:for-each> перебирает все такие элементы и обрабатывает каждый из них. Таблица стилей обрабатывает следующий документ XML:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Результат преобразования выглядит так:

```
Here is a moderately long list:
1.  The Sacred Art of Dub
2.  Only the Poor Man Feel It
3.  Excitable Boy
4.  Aki Special
5.  Combat Rock
6.  Talking Timbuktu
7.  The Birth of the Cool
```

Элемент <xsl:for-each> перебрал все элементы <listitem> в исходном документе XML и последовательно обработал каждый из них.

[2.0] <xsl:for-each-group>

Разбивает объекты последовательности на группы. Существуют четыре способа определения групп; все они подробно описаны ниже. Группировка осуществляется на основании общего значения или по шаблону, которому должен соответствовать первый или последний объект группы. Учтите, что объект исходной последовательности может принадлежать нескольким группам одновременно.

Категория

Инструкция.

Обязательный атрибут

`select`

Выражение XPath, определяющее группируемые объекты.

Необязательные атрибуты

`group-by`

Определяет значение, общее для всех объектов группы. Например, `group-by="state"` создает группу для каждого уникального значения элемента `<state>`. Все объекты одной группы обладают одинаковым значением `<state>`.

`group-adjacent`

Определяет выражение, вычисляемое для каждого объекта последовательности. Если значение выражения для текущего объекта совпадает со значением этого же выражения для предыдущего объекта, то текущий объект помещается в одну группу с предыдущим объектом. В противном случае текущий объект становится первым объектом новой группы.

`group-starting-with`

Определяет шаблон, обозначающий начало новой группы. При обнаружении объекта, соответствующего этому шаблону, начинается новая группа, а все последующие объекты включаются в эту группу, пока не будет найден следующий совпадающий объект.

`group-ending-with`

Определяет шаблон, обозначающий конец текущей группы. При обнаружении объекта, соответствующего этому шаблону, текущая группа закрывается. Следующий объект последовательности открывает новую группу.

Помните, что режимы `group-by`, `group-adjacent`, `group-starting-with` и `group-ending-with` являются взаимоисключающими.

`collation`

Определяет последовательность упорядочения, используемую для сравнения ключей группировки. Способ определения последовательностей упорядочения зависит от конкретного процессора XSLT; за подробностями обращайтесь к документации своего процессора.

Атрибут `collation` может использоваться только с атрибутами `group-by` или `group-adjacent`. Попытка его использования с `group-starting-with` или `group-ending-with` приводит к ошибке.

Содержимое

Нуль или более элементов <xsl:sort> и конструктор последовательности.

Местонахождение

Любой конструктор последовательности.

Где определяется

XSLT, раздел 14 «Grouping».

Пример

Мы рассмотрим четыре примера, демонстрирующих все четыре разновидности группировки. Начнем с group-by, самого распространенного типа группировки. В качестве исходного документа XML будет использоваться упрощенный список адресов клиентов:

```
<?xml version="1.0"?>
<!-- simplified-names.xml -->
<addressbook>
  <address>
    <first-name>Chester Hasbrouck</first-name>
    <last-name>Frisby</last-name>
    <city>Sheboygan</city>
    <state>WI</state>
  </address>
  <address>
    <first-name>Mary</first-name>
    <last-name>Backstayge</last-name>
    <city>Skunk Haven</city>
    <state>MA</state>
  </address>
  <address>
    <first-name>Natalie</first-name>
    <last-name>Attired</last-name>
    <city>Winter Harbor</city>
    <state>ME</state>
  </address>
  <address>
    <first-name>Harry</first-name>
    <last-name>Backstayge</last-name>
    <city>Skunk Haven</city>
    <state>MA</state>
  </address>
  <address>
    <first-name>Mary</first-name>
    <last-name>McGoon</last-name>
    <city>Boylston</city>
    <state>VA</state>
  </address>
  <address>
    <first-name>Amanda</first-name>
    <last-name>Reckonwith</last-name>
    <city>Lynn</city>
    <state>MA</state>
  </address>
</addressbook>
```

Мы воспользуемся режимом `group-by` для группировки адресов по штату (`state`). Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- for-each-group1.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Customers grouped by state&#xA;&#xA;</xsl:text>
    <xsl:for-each-group select="/addressbook/address" group-by="state">
      <xsl:sort select="state"/>
      <xsl:text>  State = </xsl:text>
      <xsl:value-of select="current-grouping-key()"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:for-each select="current-group()">
        <xsl:text>&#x9;</xsl:text>
        <xsl:value-of select="(first-name, last-name)"
          separator=" "/>
        <xsl:text>, </xsl:text>
        <xsl:value-of select="city"/>
        <xsl:text>&#xA;</xsl:text>
      </xsl:for-each>
    </xsl:for-each-group>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Customers grouped by state

  State = MA
        Mary Backstayge, Skunk Haven
        Harry Backstayge, Skunk Haven
        Amanda Reckonwith, Lynn
  State = ME
        Natalie Attired, Winter Harbor
  State = VA
        Mary McGoon, Boylston
  State = WI
        Chester Hasbrouck Frisby, Sheboygan
```

Элемент `<xsl:sort>`, следующий за элементом `<xsl:for-each-group>`, сортирует сами группы. Для каждой группы выводится заголовок, а затем содержимое каждой группы обрабатывается элементом `<xsl:for-each select="current-group()">`. Если убрать элемент `<xsl:sort>`, упорядочивающий группы, результат изменится. Правильная группировка клиентов сохранится, но сами группы будут следовать в другом порядке:

```
Customers grouped by state

  State = WI
        Chester Hasbrouck Frisby, Sheboygan
  State = MA
        Mary Backstayge, Skunk Haven
        Harry Backstayge, Skunk Haven
        Amanda Reckonwith, Lynn
```

```
State = ME
      Natalie Attired, Winter Harbor
State = VA
      Mary McGoon, Boylston
```

Так как группы не были отсортированы, они следуют в том порядке, в котором таблица стилей впервые обнаруживает каждый штат.

Для режима `group-adjacent` определяется выражение, которое возвращает некоторое значение для каждого объекта в последовательности. Чтобы не усложнять пример, мы воспользуемся `group-adjacent` с выражением, возвращающим `true()` или `false()`. Все смежные объекты последовательности, для которых выражение возвращает одинаковые значения, помещаются в одну группу. В данном примере в документе HTML выделяются группы смежных элементов `<p>`; они преобразуются в элементы `<ul>`, в которых каждый элемент `<p>` становится пунктом списка. Исходный документ HTML:

```
<?xml version="1.0"?>
<!-- grouping-input.html -->
<html>
  <body>
    <!-- Here's some sample text from the chapter "Sorting
         and Grouping". -->
    <h1>Steps for grouping in the Muench method</h1>
    <p>Define a <code>key</code> for the property we want
    to use for grouping.</p>
    <p>Select all of the nodes ...</p>
    <p>For each unique grouping value, ...</p>
    <h1>Steps for grouping in XSLT 2.0</h1>
    <p>Define an XPath expression ...</p>
    <p>Select all of the nodes we want to group ...</p>
    <p>Instead of dealing with each ...</p>
  </body>
</html>
```

Группировка выполняется следующей таблицей стилей:

```
<?xml version="1.0"?>
<!-- for-each-group2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" include-content-type="no"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Grouping with group-adjacent</title>
      </head>
      <body>
        <xsl:for-each-group select="html/body/*"
          group-adjacent="boolean(self::p)">
          <xsl:choose>
            <xsl:when test="current-grouping-key()">
              <ul>
                <xsl:for-each select="current-group()">
                  <li><xsl:apply-templates select="*|text()"/></li>
                </xsl:for-each>
```

```
            </ul>
          </xsl:when>
          <xsl:otherwise>
            <xsl:apply-templates select="current-group()" />
          </xsl:otherwise>
        </xsl:choose>
      </xsl:for-each-group>
    </body>
  </html>
</xsl:template>

<xsl:template match="*">
  <xsl:copy>
    <xsl:for-each select="@*">
      <xsl:copy/>
    </xsl:for-each>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
<html>
   <head>
      <title>Grouping with group-adjacent</title>
   </head>
   <body>
      <h1>Steps for grouping in the Muench method</h1>
      <ul>
         <li>Define a <code>key</code> for the property we want
            to use for grouping.
         </li>
         <li>Select all of the nodes ...</li>
         <li>For each unique grouping value, ...</li>
      </ul>
      <h1>Steps for grouping in XSLT 2.0</h1>
      <ul>
         <li>Define an XPath expression ...</li>
         <li>Select all of the nodes we want to group ...</li>
         <li>Instead of dealing with each ...</li>
      </ul>
   </body>
</html>
```

Атрибут `group-adjacent` определяет выражение XPath, возвращающее значение. В данном случае для любого элемента `<p>` выражение возвращает `true()`, а для всего остального возвращается `false()`. Впрочем, `group-adjacent` может содержать гораздо более сложное выражение. За подробностями обращайтесь к разделу «Второй тип группировки: group-adjacent» главы 7.

В третьем примере используется режим группировки `group-starting-with`. В этом примере каждый элемент `<h1>` открывает новую группу; все элементы вплоть до следующего элемента `<h1>` включаются в эту группу. Мы создадим выходной документ в формате DocBook, в котором `<h1>` и все, что следует за ним, оформляется в элемент `<sect1>`. (В этом примере снова используется документ HTML из примера `group-adjacent`.) Таблица стилей для группировки `group-starting-with`:

```
<?xml version="1.0"?>
<!-- for-each-group3.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="/">
    <chapter>
      <title>Grouping in XSLT</title>
      <xsl:apply-templates select="html/body"/>
    </chapter>
  </xsl:template>

  <xsl:template match="body">
    <xsl:for-each-group select="*" group-starting-with="h1">
      <sect1>
        <xsl:apply-templates select="current-group()"/>
      </sect1>
    </xsl:for-each-group>
  </xsl:template>

  <xsl:template match="h1">
    <title>
      <xsl:apply-templates/>
    </title>
  </xsl:template>

  <xsl:template match="p">
    <para>
      <xsl:apply-templates/>
    </para>
  </xsl:template>
  <xsl:template match="*">
    <xsl:copy>
      <xsl:for-each select="@*">
        <xsl:copy/>
      </xsl:for-each>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Наша таблица стилей генерирует документ DocBook, в котором каждый элемент `<h1>` **со всем содержимым преобразуется в элемент** `<sect1>`**:**

```
<?xml version="1.0" encoding="UTF-8"?>
<chapter>
   <title>Grouping in XSLT</title>
   <sect1>
      <title>Steps for grouping in the Muench method</title>
      <para>Define a <code>key</code> for the property we want
    to use for grouping.</para>
      <para>Select all of the nodes ...</para>
      <para>For each unique grouping value, ...</para>
   </sect1>
   <sect1>
      <title>Steps for grouping in XSLT 2.0</title>
```

```
      <para>Define an XPath expression ...</para>
      <para>Select all of the nodes we want to group ...</para>
      <para>Instead of dealing with each ...</para>
    </sect1>
</chapter>
```

Текст элемента HTML `<h1>` преобразуется в элемент DocBook `<title>`, а каждый элемент HTML `<p>` становится элементом DocBook `<para>`.

Последний пример демонстрирует группировку `group-ending-with`. Мы возьмем список элементов и разместим их в три столбца. Группа завершается каждым третьим элементом (`position() mod 3 = 0`). Исходные данные берутся из списка машин:

```
<?xml version="1.0"?>
<!-- carlist.xml -->
<cars>
  <make>Alfa Romeo</make>
  <make>Bentley</make>
  <make>Chevrolet</make>
  <make>Dodge</make>
  <make>Eagle</make>
  <make>Ford</make>
  <make>GMC</make>
  <make>Honda</make>
  <make>Isuzu</make>
  <model>Javelin</model>
  <model>K-Car</model>
  <make>Lincoln</make>
  <make>Mercedes</make>
  <make>Nash</make>
  <make>Opel</make>
  <make>Pontiac</make>
  <model>Quantum</model>
  <model>Rambler</model>
  <make>Studebaker</make>
</cars>
```

Таблица стилей:

```
<?xml version="1.0"?>
<!-- for-each-group4.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  exclude-result-prefixes="xs">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Car Makes and Models</title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>Car Makes and Models</h1>
        <p>Here are the car makes and models in
        our input document.</p>
        <table border="1" cellpadding="5">
          <xsl:apply-templates select="cars"/>
```

```
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="cars">
    <xsl:for-each-group select="make|model"
      group-ending-with="*[position() mod 3 = 0]">
      <tr>
        <xsl:choose>
          <xsl:when test="count(current-group()) = 3">
            <xsl:for-each select="current-group()">
              <xsl:apply-templates select="."/>
            </xsl:for-each>
          </xsl:when>
          <xsl:when test="count(current-group()) = 2">
            <xsl:apply-templates select="current-group()[1]"/>
            <xsl:apply-templates select="current-group()[2]"/>
            <td bgcolor="#CCCCCC">
              &#x20;&#x20;
            </td>
          </xsl:when>
          <xsl:otherwise>
            <xsl:apply-templates select="current-group()[1]"/>
            <td bgcolor="#CCCCCC" colspan="2">
              &#x20;&#x20;
            </td>
          </xsl:otherwise>
        </xsl:choose>
      </tr>
    </xsl:for-each-group>
  </xsl:template>

  <xsl:template match="make">
    <td style="font-weight: bold;">
      <xsl:apply-templates/>
    </td>
  </xsl:template>

  <xsl:template match="model">
    <td style="font-style: italic; font-weight: bold;">
      <xsl:apply-templates/>
    </td>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
<html>
   <head>
      <title>Car Makes and Models</title>
   </head>
   <body style="font-family: sans-serif;">
      <h1>Car Makes and Models</h1>
      <p>Here are the car makes and models in
         our input document.
      </p>
```

```
        <table border="1" cellpadding="5">
           <tr>
              <td style="font-weight: bold;">Alfa Romeo</td>
              <td style="font-weight: bold;">Bentley</td>
              <td style="font-weight: bold;">Chevrolet</td>
           </tr>
           ...
           <tr>
              <td style="font-weight: bold;">Studebaker</td>
              <td bgcolor="#CCCCCC" colspan="2"></td>
           </tr>
        </table>
     </body>
  </html>
```

Элементы списка машин записываются в таблицу HTML группами по три. Обратите внимание: последняя группа содержит только один элемент, поэтому за ячейкой последнего элемента следует пустая серая ячейка с атрибутом `colspan="2"`. Внешний вид документа показан на рис. А.9.

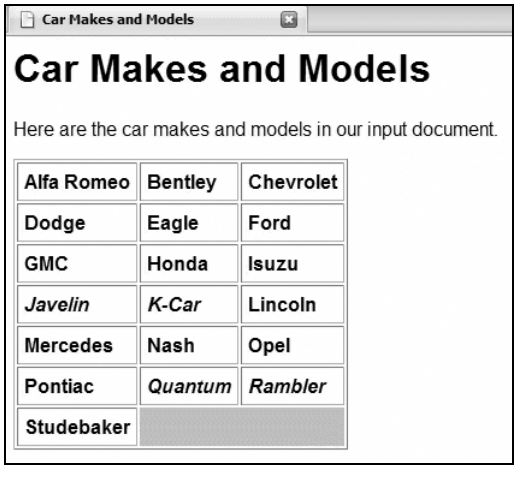


Рис. А.9. *Группировка элементов в режиме group-ending-with*

[2.0] <xsl:function>

Определяет функцию, которая может использоваться в выражениях XPath в таблице стилей.

Категория

Объявление.

Обязательный атрибут

`name`

Определяет имя функции.

Необязательные атрибуты

as

Определяет тип данных, возвращаемый функцией.

override

Определяет, должна ли функция замещать другую функцию с теми же именем и количеством аргументов, которая предоставляется процессором за пределами области видимости таблицы стилей. Значение по умолчанию override="yes" означает, что использоваться должна определяемая функция (то есть она замещает внешнюю функцию); со значением override="no" используется внешняя функция.

Содержимое

Произвольное количество элементов <xsl:param>, за которыми следует конструктор последовательности. Создаваемая последовательность определяется функцией.

Где определяется

XSLT, раздел 10.3 «Stylesheet Functions».

Пример

Таблица стилей использует три элемента <xsl:function> для получения цвета фона, размера шрифта и стиля различных элементов HTML. Размещение этой логики в функции позволяет обратиться к функции из шаблона значения атрибута. Таблица стилей выглядит так:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- function.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:sample="http://www.oreilly.com/catalog/xslt"
  exclude-result-prefixes="xs sample">

  <xsl:output method="html" include-content-type="no"/>

  <xsl:variable name="colors" as="xs:string *"
    select="('yellow', 'white', 'blue')"/>

  <xsl:variable name="fontSizes" as="xs:integer *"
    select="(18, 24, 36)"/>

  <xsl:variable name="styles" as="xs:string *"
    select="('color: black;', 'color: black;',
             'color: white; font-weight: bold;')"/>

  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of select="/list/title"/></title>
      </head>
      <body style="font-family: sans-serif;">
        <h1 style="font-size: 48;"><xsl:value-of select="/list/title"/></h1>
        <table border="3" cellpadding="5" cellspacing="5" width="50%">
          <tr>
            <xsl:for-each select="/list/listitem">
              <td style="font-size: {sample:getFontSize(position())};
```

```
                    {sample:getStyle(position())}"
                bgcolor="{sample:getColor(position())}">
                <xsl:value-of select="."/>
              </td>
            </xsl:for-each>
          </tr>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:function name="sample:getColor" as="xs:string">
    <xsl:param name="pos" as="xs:integer"/>
    <xsl:value-of select="$colors[($pos mod count($colors)) + 1]"/>
  </xsl:function>

  <xsl:function name="sample:getStyle" as="xs:string">
    <xsl:param name="pos" as="xs:integer"/>
    <xsl:value-of select="$styles[($pos mod count($styles)) + 1]"/>
  </xsl:function>

  <xsl:function name="sample:getFontSize" as="xs:integer">
    <xsl:param name="pos" as="xs:integer"/>
    <xsl:value-of select="$fontSizes[($pos mod count($fontSizes)) + 1]"/>
  </xsl:function>

</xsl:stylesheet>
```

Функции работают с переменными `colors` и `fontSizes`, определяемыми как последовательности строк и целых чисел соответственно. Функции `getColor()` и `getFontSize()` вызываются с позицией текущего узла; для определения возвращаемого объекта последовательности в них используется оператор `mod`.

Обратите внимание: индекс объекта, возвращаемого функциями, увеличивается на `1`. *Первый объект последовательности находится в позиции 1, а не 0*. Кроме того, чтобы упростить сопровождение кода, мы используем функцию `count()` для определения второго операнда оператора `mod`. При добавлении в последовательности новых цветов или размеров шрифтов нам не придется изменять определения функций в соответствии с новым размером.

Для преобразования списка альбомов используется следующая таблица стилей:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Сгенерированный документ HTML выглядит так:

```
<html>
   <head>
```

```
      <title>Albums I've bought recently:</title>
   </head>
   <body style="font-family: sans-serif;">
      <h1 style="font-size: 48;">Albums I've bought recently:</h1>
      <table border="3" cellpadding="5" cellspacing="5" width="50%">
         <tr>
            <td style="font-size: 24;
              color: black;" bgcolor="white">The Sacred Art of Dub</td>
            <td style="font-size: 36;
              color: white; font-weight: bold;" bgcolor="blue">
              Only the Poor Man Feel It</td>
            <td style="font-size: 18;
              color: black;" bgcolor="yellow">Excitable Boy</td>
            <td style="font-size: 24;
              color: black;" bgcolor="white">Aki Special</td>
            <td style="font-size: 36;
              color: white; font-weight: bold;" bgcolor="blue">Combat Rock</td>
            <td style="font-size: 18;
              color: black;" bgcolor="yellow">Talking Timbuktu</td>
            <td style="font-size: 24;
              color: black;" bgcolor="white">The Birth of the Cool</td>
         </tr>
      </table>
   </body>
</html>
```

Значения атрибутов `bgcolor` и `style` генерируются определенными нами функциями XSLT. При просмотре в броузере выходной документ выглядит так, как показано на рис. А.10.

***Рис. А.10.** Таблица HTML, сгенерированная с помощью функций XSLT*

Функции XSLT способны заметно упростить как сами таблицы стилей, так и их сопровождение. В нашей таблице стилей XSLT 2.0 атрибут `bgcolor` создается следующей разметкой:

```
bgcolor="{sample:getColor(position())}"
```

А вот как то же самое делается в XSLT 1.0:

```
<xsl:attribute name="bgcolor">
  <xsl:choose>
    <xsl:when test="position() mod 3 = 0">
      <xsl:text>yellow</xsl:text>
    </xsl:when>
    <xsl:when test="position() mod 3 = 1">
      <xsl:text>white</xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>blue</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:attribute>
```

<xsl:if>

Реализует конструкцию `if`. Элемент содержит атрибут `test` и шаблон XSLT. Если в результате вычисления атрибута `test` будет получено логическое значение `true`, обрабатывается шаблон XSLT. Элемент реализует только логику `if`; если вам потребуется реализовать конструкцию `if-then-else`, используйте элемент `<xsl:choose>` с одним элементом `<xsl:when>` и одним элементом `<xsl:otherwise>`.

Категория

Инструкция.

Обязательный атрибут

`test`

Содержит логическое выражение. Если результатом обработки выражения является логическое значение `true`, обрабатывается шаблон XSLT в элементе `<xsl:if>`.

Необязательные атрибуты

Нет.

Содержимое

Шаблон XSLT.

Местонахождение

Элемент `<xsl:if>` находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 9.1 «Conditional Processing with xsl:if».

[2.0] XSLT, раздел 8.1 «Conditional Processing with xsl:if».

Пример

Следующая таблица стилей демонстрирует использование элемента `<xsl:if>`:

```
<?xml version="1.0"?>
<!-- if.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;</xsl:text>
    <xsl:text>Here are the odd-numbered items from the list:</xsl:text>
    <xsl:text>&#xA;</xsl:text>
    <xsl:for-each select="list/listitem">
      <xsl:if test="(position() mod 2) = 1">
        <xsl:number format="1. "/>
        <xsl:value-of select="."/>
        <xsl:text>&#xA;</xsl:text>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Элемент <xsl:if> проверяет, является ли позиция заданного элемента <listitem> нечетным числом. Если условие выполняется, элемент записывается в дерево результата. Для тестирования таблицы стилей используется следующий документ XML:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list>
  <title>A few of my favorite albums</title>
  <listitem>A Love Supreme</listitem>
  <listitem>Beat Crazy</listitem>
  <listitem>You Could Have it So Much Better</listitem>
  <listitem>Kind of Blue</listitem>
  <listitem>London Calling</listitem>
  <listitem>Remain in Light</listitem>
  <listitem>The Joshua Tree</listitem>
  <listitem>The Indestructible Beat of Soweto</listitem>
</list>
```

Выполнив преобразование, мы получаем следующий результат:

```
Here are the odd-numbered items from the list:
1. A Love Supreme
3. You Could Have it So Much Better
5. London Calling
7. The Joshua Tree
```

<xsl:import>

Импортирует шаблоны, находящиеся в других таблицах стилей XSLT. В отличие от <xsl:include>, все шаблоны, импортируемые <xsl:import>, обладают более низким приоритетом по сравнению с шаблонами из включающей таблицы стилей. Другое различие между <xsl:include> и <xsl:import> заключается в том, что <xsl:include> может находиться в любом месте таблицы стилей, тогда как элемент <xsl:import> должен находиться только в начале.

Категория

Элемент верхнего уровня.

Обязательный атрибут

`href`

Определяет URI-адрес импортируемой таблицы стилей.

Необязательные атрибуты

Нет.

Содержимое

Нет. Элемент `<xsl:import>` является пустым элементом.

Местонахождение

`<xsl:import>` относится к элементам верхнего уровня, а его родителем может быть только элемент `<xsl:stylesheet>`.

Где определяется

[1.0] XSLT, раздел 2.6.2 «Stylesheet Import».

[2.0] XSLT, раздел 3.10.3 «Stylesheet Import».

Пример

Простая таблица стилей, которая будет импортироваться элементом `<xsl:import>`:

```
<?xml version="1.0"?>
<!-- listitem.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;</xsl:text>
    <xsl:apply-templates select="list/title"/>
    <xsl:apply-templates select="list/listitem"/>
  </xsl:template>

  <xsl:template match="title">
    <xsl:value-of select="."/>
    <xsl:text>: </xsl:text>
    <xsl:text>&#xA;</xsl:text>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

  <xsl:template match="listitem">
    <xsl:text>HERE IS LISTITEM NUMBER </xsl:text>
    <xsl:value-of select="position()"/>
    <xsl:text>:  </xsl:text>
    <xsl:value-of select="."/>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Документ XML, используемый при тестировании:

```
<?xml version="1.0"?>
<!-- albums.xml -->
```

```
<list>
  <title>A few of my favorite albums</title>
  <listitem>A Love Supreme</listitem>
  <listitem>Beat Crazy</listitem>
  <listitem>You Could Have it So Much Better</listitem>
  <listitem>Kind of Blue</listitem>
  <listitem>London Calling</listitem>
  <listitem>Remain in Light</listitem>
  <listitem>The Joshua Tree</listitem>
  <listitem>The Indestructible Beat of Soweto</listitem>
</list>
```

При обработке исходного документа XML с использованием первой таблицы стилей будет получен следующий результат:

```
A few of my favorite albums:

HERE IS LISTITEM NUMBER 1:  A Love Supreme
HERE IS LISTITEM NUMBER 2:  Beat Crazy
HERE IS LISTITEM NUMBER 3:  You Could Have it So Much Better
HERE IS LISTITEM NUMBER 4:  Kind of Blue
HERE IS LISTITEM NUMBER 5:  London Calling
HERE IS LISTITEM NUMBER 6:  Remain in Light
HERE IS LISTITEM NUMBER 7:  The Joshua Tree
HERE IS LISTITEM NUMBER 8:  The Indestructible Beat of Soweto
```

Первая таблица стилей импортируется во вторую элементом `<xsl:import>`:

```
<?xml version="1.0"?>
<!-- import.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:import href="listitem.xsl"/>

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;</xsl:text>
    <xsl:apply-templates select="list/title"/>
    <xsl:apply-templates select="list/listitem"/>
  </xsl:template>

  <xsl:template match="listitem">
    <xsl:value-of select="position()"/>
    <xsl:text>.  </xsl:text>
    <xsl:value-of select="."/>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Вторая таблица стилей генерирует следующий результат:

```
A few of my favorite albums:

1.  A Love Supreme
2.  Beat Crazy
3.  You Could Have it So Much Better
4.  Kind of Blue
```

```
5.  London Calling
6.  Remain in Light
7.  The Joshua Tree
8.  The Indestructible Beat of Soweto
```

Обратите внимание: в обеих таблицах стилей присутствует шаблон с атрибутом `match="listitem"`. Шаблон в импортированной таблице стилей обладает более низким приоритетом, поэтому он не используется. Однако только импортированная таблица стилей содержит шаблон с атрибутом `match="title"`, поэтому элемент `<title>` обрабатывается импортированной таблицей.

См. также

Описания элементов `<xsl:include>`, `<xsl:apply-imports>` и *[2.0]* `<xsl:next-match>`.

[2.0 – Schema] <xsl:import-schema>

Импортирует документ XML Schema. Импортирование и обработка схемы предшествует какой-либо обработке входных документов. Это позволяет определять типы данных и правила проверки до того, как процессор XSLT приступит к преобразованию входного документа. *Элемент поддерживается только схемосовместимыми процессорами XSLT 2.0.*

Категория

Элемент верхнего уровня.

Обязательные атрибуты

Нет.

Необязательные атрибуты

`namespace`

Пространство имен, используемое импортированной схемой. Любые шаблоны и функции XPath могут использовать это пространство имен, указывая тем самым, что элемент или значение определяется этой схемой.

`schema-location`

URI-ссылка на документ схемы. Если таблица стилей импортирует схему *и использует* ее, неудача при поиске схемы приводит к фатальной ошибке. Если схема только импортируется, но не используется, процессору XSLT разрешается проигнорировать ошибку.

Элемент `<xsl:import-schema>` может включать элемент `<xs:schema>`. Если элемент `<xsl:import-schema>` содержит `<xs:schema>` одновременно с атрибутом `schema-location`, происходит фатальная ошибка.

Содержимое

В элементе `<xsl:import-schema>` могут находиться некоторые элементы из пространства имен XSLT. Если `<xsl:import-schema>` имеет содержимое, скорее всего, это будет элемент `<xs:schema>`, который в свою очередь определяет типы данных и элементы. Как упоминалось ранее, наличие у `<xsl:import-schema>` содержимого одновременно с атрибутом `schema-location` является ошибкой.

Местонахождение

<xsl:import-schema> относится к элементам верхнего уровня, а его родителем может быть только элемент <xsl:stylesheet>.

Где определяется

[2.0] XSLT, раздел 3.14 «Importing Schema Components».

Пример

Для демонстрации применения <xsl:import-schema> будет использована схема заказов, которая уже неоднократно встречалась в примерах книги. Действительный заказ выглядит примерно так:

```
<?xml version="1.0"?>
<!-- good-po.xml -->
<purchase-order id="38292"
  xmlns="http://www.oreilly.com/xslt">
  <date year="2001" month="6" day="19"/>
  <customer id="4738" level="Platinum">
    <address type="business">
      <name>
        <title>Mr.</title>
        <first-name>Chester Hasbrouck</first-name>
        <last-name>Frisby</last-name>
      </name>
      <street>1234 Main Street</street>
      <city>Sheboygan</city>
      <state>WI</state>
      <zip>48392</zip>
    </address>
  </customer>
  <items>
    <item part-no="28392-33-TT">
      <partname>Turnip Twaddler</partname>
      <qty>3</qty>
      <price>9.95</price>
    </item>
    <item part-no="28100-38-CT">
      <partname>Clam Teaser</partname>
      <qty>7</qty>
      <price>39.95</price>
    </item>
  </items>
</purchase-order>
```

Теперь воспользуемся элементом <xsl:import-schema> для чтения файла схемы и документа XML. Он проверит, является ли документ XML экземпляром элемента <po:purchase-order>, определяемого в схеме:

```
<?xml version="1.0"?>
<!-- import-schema.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:po="http://www.oreilly.com/xslt">

  <xsl:import-schema namespace="http://www.oreilly.com/xslt"
```

```
    schema-location="po.xsd" />

  <xsl:output method="text"/>

  <xsl:template match="schema-element(po:purchase-order)">
    <xsl:text>&#xA;This is a test of the &lt;xsl:import-</xsl:text>
    <xsl:text>schema&gt; element.&#xA;&#xA;</xsl:text>
    <xsl:text>Here are all the items in this purchase </xsl:text>
    <xsl:text>order:&#xA;</xsl:text>
    <xsl:for-each select="po:items/po:item">
      <xsl:text>  * </xsl:text>
      <xsl:value-of select="po:partname"/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Обратите внимание: помимо импортирования схемы в атрибуте `match` используется проверка узла `schema-element`. Это означает, что шаблон совпадает только с элементами `<po:purchase-order>`, проверенными на действительность. Результат:

```
This is a test of the <xsl:import-schema> element.

Here are all the items in this purchase order:
  * Turnip Twaddler
  * Clam Teaser
```

В самой схеме атрибут `targetNamespace` и пространство имен по умолчанию (`xmlns=`) соответствуют атрибуту `namespace` элемента `<xsl:import-schema>` в таблице стилей. Для ссылок на элементы в заказе мы связываем префикс `po` с этим пространством имен. Каждый раз, когда требуется сослаться на элементы из пространства имен заказов, мы указываем префикс (например, `po:purchase-order`). Если URI-адреса пространств имен не совпадают, таблица стилей работать не будет.

Далее приводится альтернативная версия таблицы стилей, в которой элемент `<xsl:import-schema>` содержит саму схему. Эта версия выдает такие же результаты, что и предыдущая:

```
<?xml version="1.0"?>
<!-- import-schema2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns="http://www.oreilly.com/xslt"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:po="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:import-schema namespace="http://www.oreilly.com/xslt">
    <xs:schema
      targetNamespace="http://www.oreilly.com/xslt"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">

      <xs:element name="purchase-order">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="date"
              minOccurs="1" maxOccurs="1"/>
            <xs:element ref="customer"
              minOccurs="1" maxOccurs="1"/>
```

```
            <xs:element ref="items"
              minOccurs="1" maxOccurs="1"/>
          </xs:sequence>
          <xs:attribute name="id" type="xs:string"/>
        </xs:complexType>
      </xs:element>

      ...

    </xs:schema>
  </xsl:import-schema>

  <xsl:output method="text"/>

  <xsl:template match="schema-element(po:purchase-order)">
    <xsl:text>&#xA;This is a test of the <xsl:import-</xsl:text>
    <xsl:text>schema> element.&#xA;&#xA;</xsl:text>
    <xsl:text>Here are all the items in this purchase </xsl:text>
    <xsl:text>order:&#xA;</xsl:text>
    <xsl:for-each select="po:items/po:item">
      <xsl:text>  * </xsl:text>
      <xsl:value-of select="po:partname"/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

<xsl:include>

Включает другую таблицу стилей XSLT. Элемент позволяет разместить общие преобразования в отдельной таблице стилей, а затем в любой момент подключать шаблоны из этой таблицы. В отличие от `<xsl:import>`, все шаблоны, включаемые элементом `<xsl:include>`, обладают равным приоритетом с шаблонами включающей таблицы стилей. Другое отличие заключается в том, что `<xsl:include>` может находиться в любом месте таблицы стилей, тогда как элемент `<xsl:import>` должен находиться только в начале.

Категория

Элемент верхнего уровня.

Обязательный атрибут

`href`

URI-адрес включаемой таблицы стилей.

Необязательные атрибуты

Нет.

Содержимое

Нет. Элемент `<xsl:include>` является пустым элементом.

Местонахождение

`<xsl:include>` относится к элементам верхнего уровня, а его родителем может быть только элемент `<xsl:stylesheet>`.

Где определяется

[1.0] XSLT, раздел 2.6.1 «Stylesheet Inclusion».

[2.0] XSLT, раздел 3.10.2 «Stylesheet Inclusion».

Пример

Элемент <xsl:include> помогает разделить таблицу стилей на меньшие фрагменты, которые довольно часто оказываются более удобными для повторного использования. Пример включения внешней таблицы стилей:

```
<?xml version="1.0"?>
<!-- include.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:include href="include-stylesheet.xsl"/>

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:value-of select="/report/title"/>
    <xsl:text>&#xA;&#xA;</xsl:text>
    <xsl:for-each select="report/brand">
      <xsl:text>  </xsl:text>
      <xsl:value-of select="name"/>
      <xsl:text>:  </xsl:text>
      <xsl:value-of select="format-number(units, '##,###')"/>
      <xsl:text> bars sold. </xsl:text>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
    <xsl:text>&#xA;Total sales: </xsl:text>
    <xsl:value-of
      select="format-number(sum(/report/brand/units), '##,###')"/>
    <xsl:text> bars.&#xA;&#xA;</xsl:text>
    <xsl:value-of select="$copyright"/>
  </xsl:template>

</xsl:stylesheet>
```

Включаемая таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- included-stylesheet.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:variable name="copyright">
    <xsl:text>(C) Copyright 2007 DougCo Incorporated.</xsl:text>
  </xsl:variable>

</xsl:stylesheet>
```

Включаемая таблица стилей определяет переменную $copyright со стандартным текстом. Уведомление об авторском праве может применяться в десятках таблиц стилей; используя данную таблицу стилей, мы можем обеспечить присутствие этой информации во всех генерируемых документах. Если уведомление потребуется изменить (например, с наступлением 1 января), оно изменяется в одном месте. Такая методика очень удобна и для определения наборов свойств CSS, замена или удаление которых потребует минимальных усилий.

Результат применения таблицы стилей:

```
Chocolate bar sales

  Lindt - 27,408 bars sold.
  Callebaut - 8,203 bars sold.
  Valrhona - 22,101 bars sold.
  Perugina - 14,336 bars sold.
  Ghirardelli - 19,268 bars sold.

Total sales: 91,316 bars.

(C) Copyright 2007 DougCo Incorporated.
```

Переменная `$copyright` содержит текст, который определяется во включаемой таблице стилей.

<xsl:key>

Определяет индекс в текущем документе. Элемент определяется с тремя атрибутами: `name` (имя индекса), `match` (выражение XPath, описывающее индексируемые узлы) и `use` (выражение XPath, определяющее свойство, которое используется для создания индекса). Подобно тому как индексы повышают производительность операций с базами данных, ключи повышают производительность операций с таблицами стилей.

Категория

Элемент верхнего уровня.

Обязательные атрибуты

`name`

Определяет имя ключа.

`match`

Выражение XPath, которое определяет узлы, индексируемые по данному ключу.

`use`

Выражение XPath, которое определяет свойство индексируемых узлов, используемое для выборки узлов из индекса.

[2.0] В XSLT 2.0 этот атрибут является необязательным. Элемент `<xsl:key>` может содержать конструктор последовательности, который создает или выбирает индексируемые узлы.

Необязательные атрибуты

[1.0] Нет.

В XSLT 1.0 элемент `<xsl:key>` не имеет необязательных атрибутов.

[2.0] `collation`

Атрибут `collation` определяет последовательность упорядочения, используемую для проверки равенства двух значений ключа.

Содержимое

[1.0] Нет.

В XSLT 1.0 <xsl:key> является пустым элементом.

[2.0] Конструктор последовательности.

В XSLT 2.0 элемент <xsl:key> может содержать произвольный конструктор последовательности. Наличие у элемента <xsl:key> содержимого одновременно с атрибутом use является ошибкой.

Местонахождение

<xsl:key> относится к элементам верхнего уровня, а его родителем может быть только элемент <xsl:stylesheet>.

Где определяется

[1.0] XSLT, раздел 12.2 «Keys».

[2.0] XSLT, раздел 16.3 «Keys».

Пример

В этом примере используется упрощенный список клиентов:

```
<?xml version="1.0"?>
<!-- simplified-names.xml -->
<addressbook>
  <address>
    <first-name>Chester Hasbrouck</first-name>
    <last-name>Frisby</last-name>
    <city>Sheboygan</city>
    <state>WI</state>
  </address>
  <address>
    <first-name>Mary</first-name>
    <last-name>Backstayge</last-name>
    <city>Skunk Haven</city>
    <state>MA</state>
  </address>
  <address>
    <first-name>Natalie</first-name>
    <last-name>Attired</last-name>
    <city>Winter Harbor</city>
    <state>ME</state>
  </address>
  <address>
    <first-name>Harry</first-name>
    <last-name>Backstayge</last-name>
    <city>Skunk Haven</city>
    <state>MA</state>
  </address>
  <address>
    <first-name>Mary</first-name>
    <last-name>McGoon</last-name>
    <city>Boylston</city>
    <state>VA</state>
  </address>
```

```
  <address>
    <first-name>Amanda</first-name>
    <last-name>Reckonwith</last-name>
    <city>Lynn</city>
    <state>MA</state>
  </address>
</addressbook>
```

В таблице стилей для списка клиентов определяется ключ <xsl:key>:

```
<?xml version="1.0"?>
<!-- key.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:param name="searchState"/>

  <xsl:key name="customersByState" match="address" use="state"/>

  <xsl:template match="/">
    <xsl:text>All customers in </xsl:text>
    <xsl:value-of select="$searchState"/>
    <xsl:text>&#xA;&#xA;</xsl:text>
    <xsl:for-each select="key('customersByState', $searchState)">
      <xsl:value-of select="first-name"/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="last-name"/>
      <xsl:text>, </xsl:text>
      <xsl:value-of select="city"/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Ключ определяется с именем customersByState и используется для индексирования всех элементов <address> в нашем документе, а в качестве индексирующего свойства используется значение элемента <state>. При запуске процессора XSLT с параметром state=MA таблица стилей выдает следующий результат:

```
All customers in MA

Mary Backstayge, Skunk Haven
Harry Backstayge, Skunk Haven
Amanda Reckonwith, Lynn
```

При выборке всех клиентов из штата Висконсин (state=WI) индекс <xsl:key> возвращает другие значения:

```
All customers in WI

Chester Hasbrouck Frisby, Sheboygan
```

Наконец, отсутствие найденных значений не является ошибкой. Выборка всех клиентов из штата Гавайи (state=HI):

```
All customers in HI
```

[2.0] <xsl:matching-substring>

Определяет, что делать при совпадении регулярного выражения в строке. Регулярное выражение определяется в атрибуте `regex` элемента `<xsl:analyze-string>`, содержащего элемент `<xsl:matching-substring>`.

Категория

Инструкция (фактически является частью элемента `<xsl:analyze-string>`).

Обязательные атрибуты

Нет.

Необязательные атрибуты

Нет.

Содержимое

Конструктор последовательности.

Местонахождение

Элемент `<xsl:analyze-string>`.

Где определяется

XSLT, раздел 15 «Regular Expressions».

Пример

В примере использования элемента `<xsl:analyze-string>` регулярное выражение преобразует телефонные номера в формате `999-999-9999` в номера в формате `+1 (999) 999-9999`:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- analyze-string1.xsl -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:for-each select="phonelist/phonenumber">
      <xsl:analyze-string select="."
        regex="([0-9]{{3}})-([0-9]{{3}})-([0-9]{{4}})">
        <xsl:matching-substring>
          <xsl:text>&#xA;+1 (</xsl:text>
          <xsl:value-of select="regex-group(1)"/>
          <xsl:text>) </xsl:text>
          <xsl:value-of select="regex-group(2)"/>
          <xsl:text>-</xsl:text>
          <xsl:value-of select="regex-group(3)"/>
        </xsl:matching-substring>
      </xsl:analyze-string>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

В таблице стилей используется регулярное выражение `([0-9]{{3}})-(\p{{Nd}}{{3}})-([0-9]{{4}})`. Оно создает три группы, заключенные в круглые скобки. (Каждая пара скобок соответствует одной группе.) Таблица стилей применяется к списку телефонных номеров:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- phonelist.xml -->
<phonelist>
  <phonenumber>919-555-1212</phonenumber>
  <phonenumber>(919) 555-1212</phonenumber>
  <phonenumber>212.555.1212</phonenumber>
  <phonenumber>617-555-1212</phonenumber>
  <phonenumber>+86 555-1212</phonenumber>
</phonelist>
```

Результат выглядит так:

```
+1 (919) 555-1212
+1 (617) 555-1212
```

Из пяти телефонных номеров исходного документа только два совпали с регулярным выражением. В рассмотренном примере `<xsl:analyze-string>` содержит только элемент `<xsl:matching-substring>`, но элемент `<xsl:analyze-string>` может также содержать элемент `<xsl:non-matching-substring>`.

См. также

Элементы *[2.0]* `<xsl:analyze-string>` и *[2.0]* `<xsl:non-matching-substring>`.

<xsl:message>

Отправляет сообщение. Способ отправки сообщения зависит от конкретного процессора XSLT, но чаще всего сообщения записываются на стандартное устройство вывода. Элемент обычно используется в процессе отладки таблиц стилей.

Категория

Инструкция.

Обязательные атрибуты

Нет.

Необязательные атрибуты

`terminate`

Если атрибут содержит значение `yes`, процессор XSLT прерывает выполнение после выдачи сообщения. По умолчанию используется значение `no`; если `<xsl:message>` не прерывает работу процессора, то обработка продолжается после отправки сообщения.

[2.0] В XSLT 2.0 значение атрибута `terminate` может быть шаблоном значения атрибута, то есть может вычисляться на стадии выполнения.

[2.0] `select`

Выражение XPath, определяющее содержимое сообщения.

Содержимое

Шаблон XSLT.

Местонахождение

Элемент <xsl:message> находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 13 «Messages».

[2.0] XSLT, раздел 17 «Messages».

Пример

Следующая таблица стилей использует элемент <xsl:message> для трассировки преобразования документа XML. Мы снова воспользуемся документом XML со списком недавно приобретенных альбомов:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Пусть названия альбомов перечисляются в таблице HTML с чередованием цвета фона строк. Элементы <xsl:message> используются для вывода сведений о цвете фона каждой строки в таблице:

```
<?xml version="1.0"?>
<!-- message.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>
          <xsl:value-of select="list/title"/>
        </title>
      </head>
      <body style="font-family: sans-serif; color: white;">
        <h1 style="color: black;">
          <xsl:value-of select="list/title"/>
        </h1>
        <table border="1" cellpadding="5"
          style="font-weight: bold;">
          <xsl:for-each select="list/listitem">
            <tr>
              <td>
```

```
                <xsl:attribute name="style">
                  <xsl:choose>
                    <xsl:when test="position() mod 4 = 0">
                      <xsl:text>background: yellow; color: black;</xsl:text>
                      <xsl:message terminate="no">
                        <xsl:text>Background color is yellow</xsl:text>
                      </xsl:message>
                    </xsl:when>
                    <xsl:when test="position() mod 4 = 1">
                      <xsl:text>background: blue;</xsl:text>
                      <xsl:message terminate="no">
                        <xsl:text>Background color is blue</xsl:text>
                      </xsl:message>
                    </xsl:when>
                    <xsl:when test="position() mod 4 = 2">
                      <xsl:text>background: white; color: black;</xsl:text>
                      <xsl:message terminate="no">
                        <xsl:text>Background color is white</xsl:text>
                      </xsl:message>
                    </xsl:when>
                    <xsl:otherwise>
                      <xsl:text>background: black;</xsl:text>
                      <xsl:message terminate="no">
                        <xsl:text>Background color is black</xsl:text>
                      </xsl:message>
                    </xsl:otherwise>
                  </xsl:choose>
                </xsl:attribute>
                <xsl:value-of select="."/>
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Помните, что способ выдачи сообщений в спецификации XSLT не определяется. При обработке этой таблицы стилей в Saxon 8.8J вы получите следующие результаты:

```
Background color is blue
Background color is white
Background color is black
Background color is yellow
Background color is blue
Background color is white
Background color is black
```

Xalan выводит те же сообщения, но с дополнительной информацией о том, в какой строке таблицы стилей они были сгенерированы:

```
file:///c:/message.xsl; Line #34; Column #51; Background color is blue
file:///c:/message.xsl; Line #40; Column #51; Background color is white
file:///c:/message.xsl; Line #46; Column #51; Background color is black
file:///c:/message.xsl; Line #28; Column #51; Background color is yellow
```

```
file:///c:/message.xsl; Line #34; Column #51; Background color is blue
file:///c:/message.xsl; Line #40; Column #51; Background color is white
file:///c:/message.xsl; Line #46; Column #51; Background color is black
```

[2.0] <xsl:namespace>

Создает узел пространства имен в дереве результата.

Категория

Инструкция.

Обязательный атрибут

`name`

Префикс пространства имен.

Необязательный атрибут

`select`

Выражение XPath, определяющее значение самого пространства имен. Если атрибут `select` не задан, элемент `<xsl:namespace>` должен иметь содержимое.

Содержимое

Элемент `<xsl:namespace>` должен иметь либо содержимое, либо атрибут `select`; их одновременное присутствие (или отсутствие) является ошибкой. Кроме того, ошибка происходит и в том случае, если результат обработки `<xsl:namespace>` представляет собой строку нулевой длины.

Местонахождение

Элемент `<xsl:namespace>` находится в шаблоне.

Где определяется

[2.0] XSLT, раздел 11.7 «Creating Namespace Nodes».

Пример

Следующий пример позаимствован из спецификации XSLT 2.0. Таблица стилей содержит литеральный элемент результата, в котором присутствует элемент `<xsl:namespace>`:

```
<?xml version="1.0"?>
<!-- namespace.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

 <xsl:output method="xml" indent="yes"/>

  <xsl:template match="/">
    <data xsi:type="xs:integer"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <xsl:namespace name="xs"
        select="'http://www.w3.org/2001/XMLSchema'"/>
      <xsl:text>42</xsl:text>
    </data>
```

```
  </xsl:template>

</xsl:stylesheet>
```

Обратите внимание на апострофы, в которые заключено значение атрибута `select` (и вместе с которыми оно заключается во внешние кавычки); без апострофов выражение выберет все элементы `<http://www.w3.org/2001/XMLSchema>`, что приведет к ошибке.

Результат применения таблицы стилей выглядит так:

```
<?xml version="1.0" encoding="UTF-8"?>
<data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xsi:type="xs:integer">42</data>
```

Можно также записать таблицу стилей без атрибута `select`, поместив значение пространства имен в элементе `<xsl:namespace>`:

```
<?xml version="1.0"?>
<!-- namespace2.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="/">
    <data xsi:type="xs:integer"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <xsl:namespace name="xs">
        <xsl:text>http://www.w3.org/2001/XMLSchema</xsl:text>
      </xsl:namespace>
      <xsl:text>42</xsl:text>
    </data>
  </xsl:template>

</xsl:stylesheet>
```

Наконец, тот же результат можно сгенерировать при помощи таблицы стилей, в которой `<xsl:namespace>` вообще не используется:

```
<?xml version="1.0"?>
<!-- namespace3.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="/">
    <data xsi:type="xs:integer"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xsl:text>42</xsl:text>
    </data>
  </xsl:template>

</xsl:stylesheet>
```

Скорее всего, в ваших таблицах стилей элемент `<xsl:namespace>` будет использоваться только в одной ситуации: при создании нового элемента `<xsl:element>` с определением пространства имен, особенно если это пространство имен используется в содержимом. (Если пространство имен используется в именах элементов или атрибутах, оно будет скопировано в итоговый элемент автоматически.)

<xsl:namespace-alias>

Определяет псевдоним для пространств имен, прямое использование которых усложняет обработку. Элемент используется редко, но с его помощью проще всего создать таблицу стилей, генерирующую другую таблицу стилей.

Категория

Элемент верхнего уровня.

Обязательные атрибуты

`result-prefix`

Определяет префикс пространства имен, ассоциированного с псевдонимом. Префикс должен объявляться в таблице стилей независимо от того, используется ли он какими-либо элементами таблицы.

`stylesheet-prefix`

Определяет префикс, используемый в таблице стилей для ссылок на пространство имен.

[2.0] В XSLT 2.0 атрибуты `result-prefix` и `stylesheet-prefix` могут принимать значение `#default`. Естественно, это приведет к ошибке при отсутствии пространства имен по умолчанию. (Пространство имен по умолчанию определяется записью `xmlns=`.)

Необязательные атрибуты

Нет.

Содержимое

Нет. `<xsl:namespace-alias>` является пустым элементом.

Местонахождение

`<xsl:namespace-alias>` относится к элементам верхнего уровня, а его родителем может быть только элемент `<xsl:stylesheet>`.

Где определяется

[1.0] XSLT, раздел 7.1.1 «Literal Result Elements».

[2.0] XSLT, раздел 11.1.4 «Namespace Aliasing».

Пример

Как упоминалось ранее, этот элемент обычно используется при создании таблиц стилей XSLT, генерирующих другие таблицы стилей. Используйте `<xsl:namespace-alias>` в тех случаях, когда вывод должен содержать элемент или атрибут, обычно обрабатываемый процессором XSLT.

В следующем примере создается таблица стилей, которая генерирует другую таблицу стилей для копирования исходного документа в выходной поток. Исходная таблица стилей с использованием `<xsl:namespace-alias>`:

```
<?xml version="1.0"?>
<!-- namespace-alias.xsl -->
<xsl:stylesheet version="1.0"
```

```
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xslout="[anything but the XSL namespace]">

    <xsl:output method="xml" indent="yes"/>

    <xsl:namespace-alias stylesheet-prefix="xslout"
      result-prefix="xsl"/>

    <xsl:template match="/">
      <xslout:stylesheet version="1.0">
        <xslout:output method="xml"/>
        <xslout:template match="/">
          <xslout:copy-of select="."/>
        </xslout:template>
      </xslout:stylesheet>
    </xsl:template>

</xsl:stylesheet>
```

При обработке этой таблицы стилей с произвольным документом XML создается новая таблица стилей:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="xml"/>
   <xsl:template match="/">
      <xsl:copy-of select="."/>
   </xsl:template>
</xsl:stylesheet>
```

В исходной таблице стилей элемент `<xsl:namespace-alias>` используется для того, чтобы процессор XSLT не интерпретировал элементы XSLT, копируемые в выходную таблицу стилей, как инструкции. При определении пространства имен, используемого в качестве псевдонима, это пространство имен *не может* совпадать с пространством имен XSLT. В противном случае процессор XSLT обработает элементы `<xslout:` как элементы XSLT, а это будет противоречить исходной цели использования `<xsl:namespace-alias>`.

В сгенерированной таблице стилей в качестве значения пространства имен используется пространство имен, связанное с префиксом `result-prefix` в элементе `<xsl:namespace-alias>`. Учтите, что в сгенерированной таблице стилей может и не использоваться префикс `xsl`, как в нашем примере. Saxon генерирует префикс `xsl`, тогда как Xalan использует префикс `xslout` из исходной таблицы стилей. Впрочем, выбор префикса не влияет на работу сгенерированной таблицы стилей.

[2.0] Чтобы продемонстрировать использование пространства имен по умолчанию с элементом `<xsl:namespace-alias>`, мы рассмотрим две таблицы стилей; в одной используется атрибут `result-prefix="#default"`, а в другой – атрибут `stylesheet-prefix="#default"`. Первая таблица стилей:

```
<?xml version="1.0"?>
<!-- namespace-alias2.xsl -->
<stylesheet version="1.0"
  xmlns="http://www.w3.org/1999/XSL/Transform"
  xmlns:xslout="[anything but the XSL namespace]">

  <output method="xml" indent="yes"/>

  <namespace-alias stylesheet-prefix="xslout"
```

```
    result-prefix="#default"/>
  <template match="/">
    <xslout:stylesheet version="1.0">
      <xslout:output method="xml"/>
      <xslout:template match="/">
        <xslout:copy-of select="."/>
      </xslout:template>
    </xslout:stylesheet>
  </template>

</stylesheet>
```

Таблица стилей выдает следующий результат:

```
<?xml version="1.0" encoding="UTF-8"?>
<stylesheet xmlns="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <output method="xml"/>
   <template match="/">
      <copy-of select="."/>
   </template>
</stylesheet>
```

Исходная и сгенерированная таблицы стилей работают, так как все элементы XSLT находятся в пространстве имен по умолчанию `http://www.w3.org/1999/XSL/Transform`**.**

Вторая таблица стилей:

```
<?xml version="1.0"?>
<!-- namespace-alias3.xsl -->
<xsl:stylesheet version="1.0"
  xmlns="[что угодно, кроме пространства имен XSL]"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" indent="yes"/>

  <xsl:namespace-alias stylesheet-prefix="#default"
    result-prefix="xsl"/>

  <xsl:template match="/">
    <stylesheet version="1.0">
      <output method="xml"/>
      <template match="/">
        <copy-of select="."/>
      </template>
    </stylesheet>
  </xsl:template>

</xsl:stylesheet>
```

Выходные данные этой таблицы стилей выглядят более типично:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="xml"/>
   <xsl:template match="/">
      <xsl:copy-of select="."/>
   </xsl:template>
</xsl:stylesheet>
```

[2.0] <xsl:next-match>

Приказывает процессору XSLT выбрать шаблон с приоритетом на один уровень ниже, чем у текущего шаблона. Напоминает <xsl:apply-imports>, однако <xsl:next-match> работает со всеми шаблонами, включая шаблоны из главной таблицы стилей и из всех включаемых и импортируемых таблиц. При помощи <xsl:next-match> можно строить шаблоны, действующие по аналогии с переопределением методов в объектно-ориентированных языках. На концептуальном уровне работу <xsl:next-match> можно сравнить с вызовом super() в программе на языке Java.

Категория

Инструкция.

Обязательные атрибуты

Нет.

Необязательные атрибуты

Нет.

Содержимое

Произвольное количество элементов <xsl:with-param> и <xsl:fallback>.

Местонахождение

Элемент <xsl:next-match> находится в шаблоне.

Где определяется

[2.0] XSLT, раздел 6.7 «Overriding Template Rules».

Пример

В этом примере создаются шаблоны для обработки следующего документа HTML:

```
<!-- element-discussion.html -->
<html>
  <head>
    <title>Interesting new XSLT elements</title>
  </head>
  <body>
    <h1>Interesting new XSLT elements</h1>
    <p>XSLT 2.0 has lots of interesting new elements.
    We'll mention a couple of them here. </p>
    <h1>The <code>&lt;xsl:next-match&gt;</code> element</h1>
    <p>One of the most interesting new elements in XSLT 2.0
    is <code>&lt;xsl:next-match&gt;</code>. </p>
    <h1>The <code>&lt;xsl:perform-sort&gt;</code> element</h1>
    <p>Don't forget about <code>&lt;xsl:perform-sort&gt;</code>,
    though.  It's very interesting as well.</p>
  </body>
</html>
```

В данном документе присутствуют элементы HTML <code>. Одни из них находятся в элементах <h1>, другие – в элементах <p>. Для демонстрации применения <xsl:next-match> мы определим два разных шаблона, обрабатывающих элементы <code>. Базо-

вый шаблон выводит текст элемента моноширинным шрифтом; переопределяющий шаблон изменяет цвет элемента <code>, если он находится в элементе <h1>. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- next-match.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="html">
    <html>
      <head>
        <title>
          <xsl:value-of select="head/title"/>
        </title>
      </head>
      <body style="font-family: sans-serif;">
        <xsl:apply-templates select="body/*"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="h1/code">
    <span style="color: red;">
      <xsl:next-match/>
    </span>
  </xsl:template>

  <xsl:template match="code">
    <span style="font-family: monospace;">
      <xsl:apply-templates select="*|text()"/>
    </span>
  </xsl:template>

  <xsl:template match="h1">
    <h1>
      <xsl:apply-templates select="*|text()"/>
    </h1>
  </xsl:template>

  <xsl:template match="p">
    <p>
      <xsl:apply-templates select="*|text()"/>
    </p>
  </xsl:template>

</xsl:stylesheet>
```

Ключом к использованию <xsl:next-match> являются разные приоритеты задействованных шаблонов. В приведенном примере они определяются правилами приоритетов XSLT по умолчанию. Конкретное правило обладает более высоким приоритетом, чем более общее правило, поэтому самым высокоприоритетным правилом для элемента <code> в элементе <h1> является шаблон match="h1/code". Элемент <xsl:next-match> в шаблоне активизирует следующий шаблон в порядке приоритетов, а именно шаблон match="code".

В соответствии с нашей таблицей стилей по умолчанию для элемента <code> создается элемент с моноширинным шрифтом. Более конкретный (а следовательно, бо-

лее высокоприоритетный) шаблон создает элемент с красным цветом шрифта. Если потребуется назначить другой шрифт всем элементам <code>, достаточно изменить шаблон match="code", в котором задается свойство font-family. Результат применения таблицы стилей:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Interesting new XSLT elements</title>
   </head>
   <body style="font-family: sans-serif;">
      <h1>Interesting new XSLT elements</h1>
      <p>XSLT 2.0 has lots of interesting new elements.
         We'll mention a couple of them here.
      </p>
      <h1>The <span style="color: red;">
        <span style="font-family: monospace;">
        &lt;xsl:next-match&gt;</span>
        </span> element
      </h1>
      <p>One of the most interesting new elements in XSLT 2.0
         is <span style="font-family: monospace;">
        &lt;xsl:next-match&gt;</span>.
      </p>
      <h1>The <span style="color: red;">
        <span style="font-family: monospace;">
        &lt;xsl:perform-sort&gt;</span>
        </span> element
      </h1>
      <p>Don't forget about <span style="font-family: monospace;">
        &lt;xsl:perform-sort&gt;</span>,
        though.  It's very interesting as well.
      </p>
   </body>
</html>
```

Результат показан на рис. А.11.

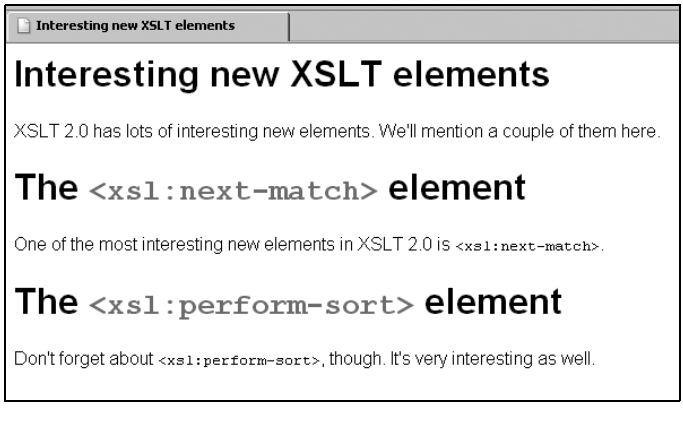


***Рис. А.11.** Код HTML, сгенерированный с использованием элемента <xsl:next-match>*

[2.0] <xsl:non-matching-substring>

Определяет, что делать при несовпадении регулярного выражения в строке. Регулярное выражение определяется в атрибуте `regex` элемента `<xsl:analyze-string>`, содержащего элемент `<xsl:non-matching-substring>`.

Категория

Инструкция (фактически является частью элемента `<xsl:analyze-string>`).

Обязательные атрибуты

Нет.

Необязательные атрибуты

Нет.

Содержимое

Конструктор последовательности.

Местонахождение

Элемент `<xsl:analyze-string>`.

Где определяется

XSLT, раздел 15 «Regular Expressions».

Пример

В следующем примере использования элемента `<xsl:analyze-string>` регулярное выражение преобразует телефонные номера в формате `999-999-9999` в номера в формате `+1 (999) 999-9999`:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- non-matching-substring.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:for-each select="phonelist/phonenumber">
      <xsl:analyze-string select="."
        regex="([0-9]{{3}})-([0-9]{{3}})-([0-9]{{4}})">
        <xsl:non-matching-substring>
          <xsl:text>&#xA;   Unrecognized phone number: </xsl:text>
          <xsl:value-of select="."/>
        </xsl:non-matching-substring>
      </xsl:analyze-string>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

В нашей таблице стилей используется регулярное выражение `([0-9]{{3}})-([0-9]{{3}})-([0-9]{{4}})`. Применение его к следующему списку телефонных номеров:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- phonelist.xml -->
<phonelist>
  <phonenumber>919-555-1212</phonenumber>
  <phonenumber>(919) 555-1212</phonenumber>
  <phonenumber>212.555.1212</phonenumber>
  <phonenumber>617-555-1212</phonenumber>
  <phonenumber>+86 555-1212</phonenumber>
</phonelist>
```

дает следующий результат:

```
Unrecognized phone number: (919) 555-1212
Unrecognized phone number: 212.555.1212
Unrecognized phone number: +86 555-1212
```

В этом примере <xsl:analyze-string> содержит только элемент <xsl:non-matching-substring>; <xsl:analyze-string> может также содержать элемент <xsl:matching-substring>.

См. также

Описание элементов *[2.0]* <xsl:analyze-string> и *[2.0]* <xsl:matching-substring>.

<xsl:number>

Выводит число. Часто используется для нумерации частей документа, хотя может также использоваться для форматирования числовых данных.

Категория

Инструкция.

Обязательные атрибуты

Нет.

Необязательные атрибуты

`count`

Атрибут `count` содержит шаблон XPath, определяющий, что именно должно подсчитываться. Если атрибут `count` не указан, подсчитываются узлы с таким же именем, как у текущего узла.

`level`

Атрибут определяет, какие уровни исходного дерева должны учитываться при нумерации элементов. Для него определены три допустимых значения: `single`, `multiple` и `any`:

`single`

Нумерация ведется только на одном уровне. Процессор XSLT переходит к первому узлу оси `ancestor-or-self`, соответствующему атрибуту `count`, а затем нумерует этот узел вместе со всеми предшествующими одноуровневыми узлами, соответствующими также атрибуту `count`. Значение используется по умолчанию.

`multiple`

Нумерация на нескольких уровнях. Процессор XSLT перебирает всех предков текущего узла вместе с самим текущим узлом и выбирает все узлы, соответствующие атрибуту `count`.

`any`

Включает всех предков текущего узла (по аналогии с `level="multiple"`), а также все элементы оси `preceding`.

Во всех этих случаях при использовании атрибута `from` анализируются только те предки, которые являются потомками ближайшего предка, соответствующего атрибуту `from`. Иначе говоря, с атрибутом `from="h1"` для нумерации анализируются только те узлы, которые находятся под ближайшим элементом `<h1>`.

`from`

Атрибут `from` содержит шаблон XPath, определяющий начальную точку нумерации. Например, `from="h1"` означает, что нумерация должна начаться с предыдущего элемента `<h1>`.

`value`

Выражение, преобразуемое в число. С помощью этого атрибута можно быстро отформатировать число; элемент `<xsl:number value="7" format="i:"/>` возвращает строку `"vii:"`.

При заданном атрибуте `value` атрибуты `count`, `level`, `from` и `select` использоваться не могут.

`format`

Атрибут `format` определяет формат генерируемой нумерации:

`format="1"`

Нумерация вида `1 2 3 4 5 6 7 8 9 10 11 ...`.

`format="01"`

Нумерация вида `01 02 03 04 ... 09 10 11 ... 99 100 101 ...`.

`format="a"`

Нумерация вида `a b c d e f ... x y z aa ab ac ...`.

`format="A"`

Нумерация вида `A B C D E F ... X Y Z AA AB AC ...`.

`format="i"`

Нумерация вида `i ii iii iv v vi vii viii ix x ...`.

`format="I"`

Нумерация вида `I II III IV V VI VII VIII IX X ...`.

[2.0] `format="w"`

Нумерация в виде словарной последовательности. С атрибутами `ordinal="yes"` и `lang="en"` нумерация форматируется в виде `first second third fourth ...`. Если атрибут `ordinal="yes"` не задан, создается последовательность вида `one two three four ...`. Если заданная комбинация `format`, `ordinal` и `lang` не поддерживается, процессор XSLT обязан прибегнуть к последнему средству – отформатировать числа в виде `1 2 3 4 5 ...`.

[2.0] `format="Ww"`

Нумерация в виде словарной последовательности. С атрибутами `ordinal="yes"` и `lang="en"` нумерация форматируется в виде `First Second Third Fourth ...`. Если атрибут `ordinal="yes"` не задан, создается последовательность вида `One Two Three Four ...`. Если заданная комбинация `format`, `ordinal` и `lang` не поддерживается, процессор XSLT обязан прибегнуть к последнему средству – отформатировать числа в виде `1 2 3 4 5 ...`.

`format="остальные обозначения"`

Зависит от используемого процессора XSLT. В спецификации XSLT перечислено несколько других схем нумерации (например, `format="&#0E51;"` для тайского языка); за информацией о поддерживаемых форматах обращайтесь к документации процессора XSLT. Если процессор XSLT не поддерживает запрашиваемую схему нумерации, спецификация XSLT требует, чтобы по умолчанию использовался атрибут `format="1"`.

[2.0] `lang`

Атрибут `lang` определяет язык, алфавит которого должен использоваться при нумерации. Разные процессоры XSLT поддерживают разные языки; за дополнительной информацией обращайтесь к документации своего процессора XSLT.

`letter-value`

Атрибут имеет допустимые значения `alphabetic` и `traditional`. Во многих языках поддерживаются две словарные схемы нумерации; в одной числовые значения представляются в алфавитной последовательности, а в другой используется традиция, специфическая для данного языка. (Одним из примеров нумерации, не использующей алфавитный порядок, служит римская нумерация.) По умолчанию используется значение `alphabetic`.

`grouping-separator`

Атрибут определяет символ, используемый для разделения групп разрядов в сгенерированном числе. По умолчанию используется запятая (`,`).

`grouping-size`

Атрибут определяет количество цифр в группах; по умолчанию используется значение `3`.

[2.0] `ordinal`

Управляет использованием порядковых числительных; допустимые значения: `yes` и `no`. Атрибуты `ordinal`, `format` и `lang` работают совместно. Например, комбинация `ordinal="yes"`, `format="1"` и `lang="en"` создает последовательность `1st 2nd 3rd 4th`. С атрибутом `format="w"` нумерация принимает вид `first second third fourth`, а с атрибутом `format="Ww"` – `First Second Third Fourth`. Если комбинация `ordinal`, `format` и `lang` не поддерживается, процессор XSLT обязан сгенерировать обычную числовую нумерацию (`1 2 3 4`). За информацией о комбинациях языков и форматов, поддерживаемых с атрибутом `ordinal`, обращайтесь к документации своего процессора XSLT.

[2.0] `select`

Выбирает нумеруемый узел. Атрибут позволяет выбрать узел, отличный от контекстного узла.

Содержимое

Нет. <xsl:number> является пустым элементом.

Местонахождение

Элемент <xsl:number> находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 7.7 «Numbering».

[2.0] XSLT, раздел 12 «Numbering».

Пример

Для полноценной демонстрации применения <xsl:number> потребуется документ XML с достаточно большим количеством элементов для нумерации. Мы используем следующий документ:

```
<?xml version="1.0"?>
<!-- items-to-number.xml -->
<book>
  <chapter><title>Alfa Romeo</title>
    <sect1><title>Bentley</title></sect1>
    <sect1><title>Chevrolet</title>
      <sect2><title>Dodge</title>
        <sect3><title>Eagle</title></sect3>
      </sect2>
    </sect1>
  </chapter>
  <chapter><title>Ford</title>
    <sect1><title>GMC</title>
      <sect2><title>Honda</title>
        <sect3><title>Isuzu</title></sect3>
        <sect3><title>Javelin</title></sect3>
        <sect3><title>K-Car</title></sect3>
        <sect3><title>Lincoln</title></sect3>
      </sect2>
      <sect2><title>Mercedes</title></sect2>
      <sect2><title>Nash</title>
        <sect3><title>Opel</title></sect3>
        <sect3><title>Pontiac</title></sect3>
      </sect2>
      <sect2><title>Quantum</title>
        <sect3><title>Rambler</title></sect3>
        <sect3><title>Studebaker</title></sect3>
      </sect2>
    </sect1>
    <sect1><title>Toyota</title></sect1>
  </chapter>
</book>
```

Элемент <xsl:number> будет использоваться несколькими разными способами, демонстрирующими различные варианты нумерации. Мы рассмотрим несколько коротких таблиц стилей и их результаты. Первая таблица:

```
<?xml version="1.0"?>
<!-- number1.xsl -->
```

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="book">
    <xsl:for-each select="chapter|.//sect1|.//sect2|.//sect3">
      <xsl:number level="multiple" count="chapter|sect1|sect2|sect3"
          format="1.1.1.1. "/>
        <xsl:value-of select="title"/>
        <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
1. Alfa Romeo
1.1. Bentley
1.2. Chevrolet
1.2.1. Dodge
1.2.1.1. Eagle
2. Ford
2.1. GMC
2.1.1. Honda
2.1.1.1. Isuzu
2.1.1.2. Javelin
2.1.1.3. K-Car
2.1.1.4. Lincoln
2.1.2. Mercedes
2.1.3. Nash
2.1.3.1. Opel
2.1.3.2. Pontiac
2.1.4. Quantum
2.1.4.1. Rambler
2.1.4.2. Studebaker
2.2. Toyota
```

В этом примере нумерация `level="multiple"` подсчитывает элементы `<chapter>`, `<sect1>`, `<sect2>` и `<sect3>`. При многоуровневой нумерации компоненты номера разделяются точками. Взглянув на номер в строке `Studebaker`, мы сразу видим, что это второй элемент `<sect3>` в четвертом элементе `<sect2>` в первом элементе `<sect1>` внутри второго элемента `<chapter>`.

В таблице стилей элемент `<xsl:for-each>` выбирает нумеруемые элементы, а элемент `<xsl:number>` создает нумерацию. Как вы вскоре увидите, для правильной нумерации всех объектов необязательно выбирать их в `<xsl:for-each>`. Если написать выражение XPath, которое выбирает только элемент `<sect3>` с `<title>Studebaker</title>`, то вывод будет иметь вид `2.1.4.2 Studebaker` (как и в нашем примере).

В следующей таблице стилей атрибут `level="any"` используется для последовательной нумерации всех элементов `<chapter>`, `<sect1>`, `<sect2>` и `<sect3>`. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- number2.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
  <xsl:output method="text"/>

  <xsl:template match="book">
    <xsl:for-each select="chapter|.//sect1|.//sect2|.//sect3">
      <xsl:number level="any" count="chapter|sect1|sect2|sect3"
        format="1. "/>
        <xsl:value-of select="title"/>
        <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Результат получается более простым, чем в предыдущем примере:

```
1. Alfa Romeo
2. Bentley
3. Chevrolet
4. Dodge
5. Eagle
6. Ford
7. GMC
8. Honda
9. Isuzu
10. Javelin
11. K-Car
12. Lincoln
13. Mercedes
14. Nash
15. Opel
16. Pontiac
17. Quantum
18. Rambler
19. Studebaker
20. Toyota
```

Таблица стилей *number3.xsl* использует `level="single"` **для раздельной нумерации элементов. Иначе говоря, четвертому элементу** `<sect3>` **внутри элемента** `<sect2>` **будет присвоен номер 4 (или** `iv`**, или** `D`**, или иное соответствующее значение). Обратите внимание: номер, присваиваемый каждому элементу, совпадает с последним компонентом номера из примера 1.**

```
<?xml version="1.0"?>
<!-- number3.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="book">
    <xsl:for-each select="chapter|.//sect1|.//sect2|.//sect3">
      <xsl:number level="single" count="chapter|sect1|sect2|sect3"
        format="1.1.1.1. "/>
        <xsl:value-of select="title"/>
        <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Результат получается относительно бесполезным:

```
1. Alfa Romeo
1. Bentley
2. Chevrolet
1. Dodge
1. Eagle
2. Ford
1. GMC
1. Honda
1. Isuzu
2. Javelin
3. K-Car
4. Lincoln
2. Mercedes
3. Nash
1. Opel
2. Pontiac
4. Quantum
1. Rambler
2. Studebaker
2. Toyota
```

В таблицу стилей *number4.xsl* вносятся кое-какие изменения. Во-первых, в ней используются алфавитная нумерация в верхнем регистре и римская нумерация в нижнем регистре. Во-вторых, элементы нумеруются на нескольких уровнях (для элементов <chapter>, <sect1> и <sect2>), но обрабатываются только элементы <sect2>. Несмотря на то что выводится только текст названий элементов <sect2>, мы все равно можем сгенерировать соответствующую многоуровневую нумерацию:

```
<?xml version="1.0"?>
<!-- number4.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="book">
    <xsl:for-each select=".//sect2">
      <xsl:number level="multiple" count="chapter|sect1|sect2"
        format="I-A-i: "/>
        <xsl:value-of select="title"/>
        <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Так как в этом случае нумеруются только элементы <sect2>, вывод становится заметно короче:

```
I-B-i: Dodge
II-A-i: Honda
II-A-ii: Mercedes
II-A-iii: Nash
II-A-iv: Quantum
```

Как упоминалось ранее, элементы, выбранные элементом <xsl:for-each>, правильно нумеруются даже в том случае, если мы не ссылаемся на соседние элементы.

Таблица стилей *number5.xsl* генерирует номера аналогично *number4.xsl*, за исключением того, что в ней используется атрибут `from`. Нумерация элементов `<sect3>` производится за четыре этапа. Сначала подсчитываются предки `<chapter>`, начиная с первого предка `<book>`, затем подсчитываются предки `<sect1>`, начиная с первого предка `<chapter>`, и т. д.:

```
<?xml version="1.0"?>
<!-- number5.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="book">
    <xsl:for-each select=".//sect3">
      <xsl:number level="any" from="book" count="chapter" format="1."/>
      <xsl:number level="any" from="chapter" count="sect1" format="1."/>
      <xsl:number level="any" from="sect1" count="sect2" format="1."/>
      <xsl:number level="any" from="sect2" count="sect3" format="1. "/>
      <xsl:value-of select="title"/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Результат сходен с тем, который выводился для таблицы *number4.xsl*:

```
1.2.1.1. Eagle
2.1.1.1. Isuzu
2.1.1.2. Javelin
2.1.1.3. K-Car
2.1.1.4. Lincoln
2.1.3.1. Opel
2.1.3.2. Pontiac
2.1.4.1. Rambler
2.1.4.2. Studebaker
```

В таблице стилей *number6.xsl* нумерация начинается с 1000; также здесь используется атрибут `grouping-separator`:

```
<?xml version="1.0"?>
<!-- number6.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="book">
    <xsl:for-each select="chapter|.//sect1|.//sect2|.//sect3">
      <xsl:variable name="value1">
        <xsl:number level="any" count="chapter|sect1|sect2|sect3"/>
      </xsl:variable>
      <xsl:number value="$value1 + 999"
        grouping-separator="," grouping-size="3"/>
      <xsl:text>. </xsl:text>
      <xsl:value-of select="title"/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
```

```
    </xsl:template>

</xsl:stylesheet>
```

Чтобы нумерация начиналась с 1000, мы сохраняем число, сгенерированное элементом <xsl:number>, в переменной. Затем элемент <xsl:number> используется повторно для вывода значения, увеличенного на 999. Результат выглядит так:

```
1,000. Alfa Romeo
1,001. Bentley
1,002. Chevrolet
1,003. Dodge
1,004. Eagle
1,005. Ford
1,006. GMC
1,007. Honda
1,008. Isuzu
1,009. Javelin
1,010. K-Car
1,011. Lincoln
1,012. Mercedes
1,013. Nash
1,014. Opel
1,015. Pontiac
1,016. Quantum
1,017. Rambler
1,018. Studebaker
1,019. Toyota
```

В таблице стилей *number7.xsl* элементы нумеруются от первого и второго элементов <sect1> (от элементов <sect1>, для которых значение position() меньше 3) во втором элементе <chapter>:

```
<?xml version="1.0"?>
<!-- number7.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="book">
    <xsl:for-each select="chapter[2]/sect1[position() < 3]">
      <xsl:for-each select="chapter|.//sect1|.//sect2|.//sect3">
        <xsl:number level="multiple" count="chapter|sect1|sect2|sect3"
          format="1.1.1.1. "/>
        <xsl:value-of select="title"/>
        <xsl:text>&#xA;</xsl:text>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Результат выглядит так:

```
2.1.1. Honda
2.1.1.1. Isuzu
2.1.1.2. Javelin
2.1.1.3. K-Car
```

```
2.1.1.4. Lincoln
2.1.2. Mercedes
2.1.3. Nash
2.1.3.1. Opel
2.1.3.2. Pontiac
2.1.4. Quantum
2.1.4.1. Rambler
2.1.4.2. Studebaker
```

Такой стиль нумерации пригодится для построения неполных сводок содержимого.

[2.0] Пора познакомиться с новыми возможностями нумерации, появившимися в XSLT 2.0. Для сокращения объема выходных данных мы будем подсчитывать только элементы `<sect2>`, как это делалось в *number4.xsl*. Начнем с использования атрибута `ordinal`:

```
<?xml version="1.0"?>
<!-- number8.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="book">
    <xsl:for-each select=".//sect2">
      <xsl:number level="any" count="chapter|sect1|sect2|sect3"
        format="Ww - " ordinal="yes"/>
        <xsl:value-of select="title"/>
        <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Словарная нумерация выглядит так:

```
Fourth - Dodge
Eighth - Honda
Thirteenth - Mercedes
Fourteenth - Nash
Seventeenth - Quantum
```

Теперь мы воспользуемся комбинацией `format` и `lang`:

```
<?xml version="1.0"?>
<!-- number9.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="book">
    <xsl:for-each select=".//sect2">
      <xsl:number level="any" count="chapter|sect1|sect2|sect3"
        format="w - " lang="de"/>
        <xsl:value-of select="title"/>
        <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

При обработке этой таблицы стилей в Saxon нумерация выводится на немецком языке:

```
vier - Dodge
acht - Honda
dreizehn - Mercedes
vierzehn - Nash
siebzehn - Quantum
```

Если запросить комбинацию атрибутов `format`, `ordinal` и `lang`, не поддерживаемую процессором XSLT, процессор возвращается к стандартному поведению. Таблица стилей *number10.xsl* показывает, что делает Saxon при запросе нумерации на польском языке:

```
<?xml version="1.0"?>
<!-- number10.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="book">
    <xsl:for-each select=".//sect2">
      <xsl:number level="any" count="chapter|sect1|sect2|sect3"
        format="w - " lang="pl"/>
        <xsl:value-of select="title"/>
        <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Saxon не поддерживает польский язык, поэтому нумерация выводится на английском:

```
four - Dodge
eight - Honda
thirteen - Mercedes
fourteen - Nash
seventeen - Quantum
```

В последнем примере используются атрибуты `format="๑"` (тайская нумерация) и `ordinal="yes"`:

```
<?xml version="1.0"?>
<!-- number11.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="book">
    <html>
      <head>
        <title>Thai numbering</title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>Thai numbering</h1>
        <p style="font-size: 150%">
          <xsl:for-each select=".//sect2">
            <xsl:number level="any" count="sect2"
```

```
                format="&#x0E51; "/>
              <xsl:value-of select="title"/>
              <br/>
            </xsl:for-each>
          </p>
        </body>
      </html>
    </xsl:template>

</xsl:stylesheet>
```

Таблица стилей генерирует документ HTML, в котором элементы <sect2> нумеруются на тайском языке:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Thai numbering</title>
   </head>
   <body style="font-family: sans-serif;">
      <h1>Thai numbering</h1>
      <p style="font-size: 150%">&#x0E51; Dodge<br>
      &#x0E52; Honda<br>&#x0E53; Mercedes<br>
      &#x0E54; Nash<br>&#x0E55; Quantum<br></p>
   </body>
</html>
```

На рис. А.12 показано, как выглядит в броузере сгенерированный документ после форматирования.

Рис. А.12. Список с тайской нумерацией

<xsl:otherwise>

Определяет логику else или default для элемента <xsl:choose>. Элемент всегда находится внутри элемента <xsl:choose> и может быть только последним из вложенных элементов.

Категория

Вложенная инструкция (<xsl:otherwise> всегда является частью элемента <xsl:choose>).

Обязательные атрибуты

Нет.

Необязательные атрибуты

Нет.

Содержимое

Шаблон.

Местонахождение

Элемент <xsl:choose>.

Где определяется

[1.0] XSLT, раздел 9.2 «Conditional Processing with xsl:choose».

[2.0] XSLT, раздел 8.2 «Conditional Processing with xsl:choose».

Пример

В следующем примере <xsl:choose> используется для выбора фонового и основного цветов в строках таблицы HTML. Таблица стилей перебирает четыре возможных значения, а элемент <xsl:otherwise> используется для определения значения по умолчанию атрибута style в генерируемом документе HTML. Таблица стилей обрабатывает следующий документ XML:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en_GB">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Таблица стилей:

```
<?xml version="1.0"?>
<!-- otherwise.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>
          <xsl:value-of select="list/title"/>
        </title>
      </head>
      <body style="font-family: sans-serif; color: white;">
        <h1 style="color: black;">
          <xsl:value-of select="list/title"/>
        </h1>
        <table border="1" cellpadding="5"
```

```
          style="font-weight: bold;">
          <xsl:for-each select="list/listitem">
            <tr>
              <td>
                <xsl:attribute name="style">
                  <xsl:choose>
                    <xsl:when test="position() mod 4 = 0">
                      <xsl:text>background: yellow; color: black;</xsl:text>
                    </xsl:when>
                    <xsl:when test="position() mod 4 = 1">
                      <xsl:text>background: blue;</xsl:text>
                    </xsl:when>
                    <xsl:when test="position() mod 4 = 2">
                      <xsl:text>background: white; color: black;</xsl:text>
                    </xsl:when>
                    <xsl:otherwise>
                      <xsl:text>background: black;</xsl:text>
                    </xsl:otherwise>
                  </xsl:choose>
                </xsl:attribute>
                <xsl:value-of select="."/>
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Сгенерированный документ HTML перебирает разные цвета фона и текста:

```
<html>
  <head>
    <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Albums I've bought recently:</title>
  </head>
  <body style="font-family: sans-serif; color: white;">
    <h1 style="color: black;">Albums I've bought recently:</h1>
    <table border="1" cellpadding="5" style="font-weight: bold;">
      <tr>
        <td style="background: blue;">The Sacred Art of Dub</td>
      </tr>
      <tr>
        <td style="background: white; color: black;">Only the Poor Man Feel It</td>
      </tr>
      <tr>
        <td style="background: black;">Excitable Boy</td>
      </tr>
      <tr>
        <td style="background: yellow; color: black;">Aki Special</td>
      </tr>
      ...
    </table>
  </body>
</html>
```

На рис. А.13 показано, как сгенерированный документ HTML выглядит в броузере.

Обратите внимание: каждой четвертой строке назначается стиль `"background:black;"`. Это значение генерируется элементом `<xsl:otherwise>`.

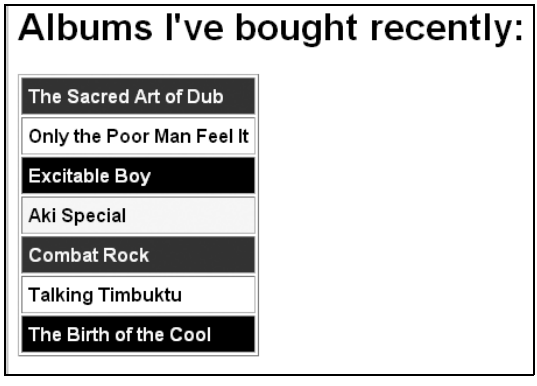


***Рис. А.13.** Использование элементов <xsl:when> и <xsl:otherwise> для перебора цветов фона*

<xsl:output>

Определяет характеристики выходного документа.

Категория

Элемент верхнего уровня.

Обязательные атрибуты

Нет.

Необязательные атрибуты

`method`

Обычно принимает одно из четырех значений: `xml`, `html`, `xhtml` или `text`. Значение определяет тип генерируемого документа. Процессор XSLT может дополнить список другими значениями; влияние этих значений на генерируемый документ определяется процессором XSLT. По умолчанию используется значение `xml`, если только корневым элементом выходного документа не является `<html>`.

[1.0] В стандарте XSLT 1.0 определяются только три метода вывода: `xml`, `html` и `text`. *[2.0]* В XSLT 2.0 добавлен метод `xhtml`, который должен поддерживаться процессором XSLT. Процессорам разрешается добавлять другие значения. Для упрощения дальнейшего описания мы будем относить `xhtml` к числу стандартных типов вывода без постоянных напоминаний о том, что в XSLT 1.0 этот метод не поддерживается.

`version`

Определяет версию выходного документа XML, HTML или XHTML. Игнорируется при наличии атрибута `method="text"`.

Атрибут version элемента <xsl:output> своим поведением отличается от атрибута version, поддерживаемого всеми элементами XSLT в XSLT 2.0. За подробностями обращайтесь к разделу «[2.0] Атрибуты, общие для всех элементов XSLT» ранее в этом приложении.

encoding

Определяет значение encoding, задаваемое в объявлении XML выходного документа. Игнорируется при наличии атрибута method="text".

omit-xml-declaration

Определяет, должно ли объявление XML быть опущено в выходном документе. Допустимые значения: yes и no; по умолчанию используется no. Атрибут используется только в режимах method="xml" и method="xhtml".

standalone

Определяет значение атрибута standalone в объявлении XML выходного документа. Допустимые значения: yes и no. Атрибут используется только в режимах method="xml" и method="xhtml".

[2.0] В XSLT 2.0 поддерживается также значение omit. Атрибут standalone="omit" означает, что в выходном документе значение standalone не задано.

doctype-public

Определяет значение атрибута PUBLIC в объявлении DOCTYPE выходного документа. Атрибут определяет открытый идентификатор DTD выходного документа. Игнорируется при наличии атрибута method="text".

doctype-system

Определяет значение атрибута SYSTEM в объявлении DOCTYPE в выходном документе. Атрибут определяет системный идентификатор DTD выходного документа. Игнорируется при наличии атрибута method="text".

cdata-section-elements

Перечисляет элементы, содержимое которых записывается в выходной документ в виде секций CDATA. Соблюдение всех ограничений и правил экранирования для секций CDATA обеспечивается процессором XSLT. Если потребуется перечислить более одного элемента, разделите их имена одним или несколькими символами-пропусками. Атрибут используется только в режимах method="xml" и method="xhtml".

indent

Определяет наличие отступов в тегах выходного документа. Допустимые значения: yes и no. Атрибут используется только в режимах method="xml", method="html" и method="xhtml", причем процессор XSLT не обязан обеспечивать его реализацию. По умолчанию используются значение yes для method="html" и method="xhtml" и значение no для method="xml".

media-type

Определяет тип MIME для выходного документа. По умолчанию используются значения text/xml для method="xml", text/html для method="html" и method="xhtml" и text/plain для method="text".

[2.0] name

Определяет имя спецификации выходного документа. Новый элемент *[2.0]* <xsl:result-document> может использовать ссылки на это имя в атрибуте format.

Если таблица стилей использует элемент <xsl:result-document> для создания нескольких выходных документов, определите спецификацию выходного документа с атрибутом name и используйте его там, где потребуется.

[2.0] byte-order-mark

Определяет, должна ли метка порядка байтов записываться в выходной документ. Допустимые значения: yes и no. По умолчанию значение yes используется для кодировки UTF-16, тогда как значение по умолчанию для UTF-8 зависит от процессора XSLT. Для всех остальных кодировок по умолчанию используется значение no.

[2.0] escape-uri-attributes

Определяет, должны ли специальные символы в атрибутах HTML и XHTML с URI-значениями заменяться своими шестнадцатеричными кодами. Например, если URI-адрес содержит пробелы, в преобразованном значении каждый пробел заменяется обозначением %20.

[2.0] include-content-type

Определяет, должен ли тип содержимого записываться в выходной документ. Допустимые значения: yes и no; по умолчанию используется значение yes. Например, при использовании атрибута include-content-type для method="html" в элемент <head> документа HTML включается элемент <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">.

[2.0] normalization-form

Определяет способ нормализации символов Юникода. Допустимые значения: NFC, NFD, NFKC, NFKD, fully-normalized и none. Процессоры XSLT также могут поддерживать другие значения; за информацией о поддержке других форм нормализации обращайтесь к документации своего процессора.

[2.0] undeclare-prefixes

Определяет, должен ли выходной документ включать отменяющие объявления пространств имен (такие объявления связывают префикс пространства имен с пустой строкой, например xmlns:doug=""). Допустимые значения: yes и no; этот атрибут имеет смысл только с method="xml" и в том случае, если атрибут version равен 1.1 или выше. За полной информацией об этом малопонятном атрибуте обращайтесь к разделу 20 спецификации XSLT 2.0.

[2.0] use-character-maps

Определяет список именованных карт символов, разделенных пробелами (карты создаются элементом *[2.0]* <xsl:character-map>, используемым при создании выходного документа).

Содержимое

Нет. <xsl:output> является пустым элементом.

Местонахождение

<xsl:output> относится к элементам верхнего уровня, а его родителем может быть только элемент <xsl:stylesheet>.

Где определяется

[1.0] XSLT, раздел 16 «Output».

[2.0] XSLT, раздел 20 «Serialization».

Пример

Для демонстрации четырех методов вывода, определяемых в спецификации XSLT, мы создадим четыре таблицы стилей – по одной для каждого метода. Во всех четырех примерах используется следующий документ XML:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Рассмотрим сами таблицы стилей и результаты, выдаваемые каждой из них. Начнем с таблицы стилей для атрибута `method="xml"`:

```
<?xml version="1.0"?>
<!-- output1.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" indent="yes" encoding="ISO-8859-1"/>

  <xsl:template match="/">
    <catalog>
      <xsl:for-each select="/list/listitem">
        <album>
          <xsl:apply-templates/>
        </album>
      </xsl:for-each>
    </catalog>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
   <album>The Sacred Art of Dub</album>
   <album>Only the Poor Man Feel It</album>
   <album>Excitable Boy</album>
   <album>Aki Special</album>
   <album>Combat Rock</album>
   <album>Talking Timbuktu</album>
   <album>The Birth of the Cool</album>
</catalog>
```

Выходной документ содержит атрибут `encoding`, указанный в нашей таблице стилей, и в выходном документе присутствуют отступы. (С атрибутом `indent="no"` генерируется синтаксически корректный (well-formed), но неудобочитаемый документ XML.)

Перейдем к версии таблицы стилей для HTML:

```
<?xml version="1.0"?>
<!-- output2.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" encoding="ISO-8859-1"/>

  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of select="/list/title"/></title>
      </head>
      <body>
        <h1><xsl:value-of select="/list/title"/></h1>
        <p>
          <xsl:for-each select="/list/listitem">
            <xsl:number format="1. "/>
            <xsl:value-of select="."/>
            <br/>
          </xsl:for-each>
        </p>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Документ HTML, сгенерированный этой таблицей:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
      <title>Albums I've bought recently:</title>
   </head>
   <body>
      <h1>Albums I've bought recently:</h1>
      <p>1. The Sacred Art of Dub<br>2. Only the Poor Man
Feel It<br>3. Excitable Boy<br>4. Aki Special<br>5.
Combat Rock<br>6. Talking Timbuktu<br>7. The Birth of the
Cool<br></p>
   </body>
</html>
```

(Чтобы листинг нормально читался, в него включены разрывы строк.) Процессор XSLT автоматически вставил элемент `<META>` **в элемент** `<head>` **документа HTML. Обратите внимание на традиционный формат тегов** `<br>`**. Формально код документа, создаваемого при таком стиле вывода XSLT, не является корректным (well-formed) кодом XML (или XHTML), однако он работает во всех существующих броузерах HTML.**

Чтобы сгенерировать корректный код XHTML, следует использовать атрибут `method="xhtml"`**:**

```
<?xml version="1.0"?>
<!-- output3.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output
```

```
    method="xhtml"
    encoding="ISO-8859-3"
    doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"/>

  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of select="/list/title"/></title>
      </head>
      <body>
        <h1><xsl:value-of select="/list/title"/></h1>
        <p>
          <xsl:for-each select="/list/listitem">
            <xsl:number format="1. "/>
            <xsl:value-of select="."/>
            <br/>
          </xsl:for-each>
        </p>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Результат выглядит так:

```
<?xml version="1.0" encoding="ISO-8859-3"?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org
/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
   <head>
      <title>Albums I've bought recently:</title>
   </head>
   <body>
      <h1>Albums I've bought recently:</h1>
      <p>1. The Sacred Art of Dub
         <br></br>2. Only the Poor Man Feel It
         <br></br>3. Excitable Boy
         <br></br>4. Aki Special
         <br></br>5. Combat Rock
         <br></br>6. Talking Timbuktu
         <br></br>7. The Birth of the Cool
         <br></br>
      </p>
   </body>
</html>
```

(В объявление `DOCTYPE` **был добавлен разрыв строки; в исходном варианте оно выводится в одной длинной строке.) Теперь элементы** `<br>` **соответствуют требованиям корректного кода XML.**

В последней таблице стилей используется атрибут `method="text"`**:**

```
<?xml version="1.0"?>
<!-- output4.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:output method="text"
  encoding="ISO-8859-3"
  indent="yes"
  omit-xml-declaration="no"
  standalone="yes"
  doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
  doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"/>

<xsl:template match="/">
  <html>
    <head>
      <title><xsl:value-of select="/list/title"/></title>
    </head>
    <body>
      <h1><xsl:value-of select="/list/title"/></h1>
      <p>
        <xsl:for-each select="/list/listitem">
          <xsl:number format="1. "/>
          <xsl:value-of select="."/>
          <br/>
        </xsl:for-each>
      </p>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Результаты, полученные при обработке таблицы стилей:

```
Albums I've bought recently:Albums I've bought recently:1. The Sacred
Art of Dub2. Only the Poor Man Feel It3. Excitable Boy4. Aki Special5.
Combat Rock6. Talking Timbuktu7. The Birth of the Cool
```

(Как и ранее, в документ были вставлены разрывы строк, чтобы листинг нормально размещался на странице.) Результат оказывается фактически бесполезным. Почему же наши тщательно закодированные элементы HTML и информация, помещенная в элемент `<xsl:output>`, не попали в текстовый документ? Дело в том, что в режиме `method="text"` в дерево результата выводятся только текстовые узлы, а большинство атрибутов `<xsl:output>` игнорируется. В ходе преобразования генерируются различные элементы HTML, однако все они игнорируются при выводе из-за атрибута `method="text"`.

[2.0] <xsl:output-character>

Определяет символ и строку, которая должна его заменять. Является аналогом объявления XML `<!ENTITY>`.

Категория

Объявление (фактически является частью элемента `<xsl:character-map>`).

Обязательные атрибуты

`character`

Заменяемый символ.

`string`

Строка, заменяющая символ.

Необязательные атрибуты

Нет.

Содержимое

Нет. `<xsl:output-character>` является пустым элементом.

Местонахождение

Элемент `<xsl:character-map>`.

Где определяется

XSLT, раздел 20 «Serialization».

Пример

Далее приводится небольшой фрагмент примера из описания элемента `<xsl:character-map>`. Эта таблица стилей заменяет два символа с цифрами в кружках (символы Юникода ૜ и ૝) графикой:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- character-map2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" use-character-maps="circles"/>

  <xsl:character-map name="circles">
    <xsl:output-character character="&#x2780;"
      string="&lt;img src='images/circle1.gif'
      width='28' height='28'/&gt;"/>
    <xsl:output-character character="&#x2781;"
      string="&lt;img src='images/circle2.gif'
      width='28' height='28'/&gt;"/>
  </xsl:character-map>

  <xsl:template match="char-test">
    <html>
      <head>
        <title>A test of some special characters</title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>A test of some special characters</h1>
        <xsl:apply-templates select="*"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="special-char">
    <p style="font-size: 200%;">
      <xsl:text>Here's a special character: </xsl:text>
      <xsl:value-of select="."/>
    </p>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей используется для преобразования следующего документа:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- special-characters2.xml -->
<char-test>
  <special-char>&#x2780;</special-char>
  <special-char>&#x2781;</special-char>
</char-test>
```

В сгенерированном документе цифры в кружках заменяются графикой:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>A test of some special characters</title>
   </head>
   <body style="font-family: sans-serif;">
      <h1>A test of some special characters</h1>
      <p style="font-size: 200%;">Here's a special character:
      <img src='images/circle1.gif' width='28' height='28'/></p>
      <p style="font-size: 200%;">Here's a special character:
      <img src='images/circle2.gif' width='28' height='28'/></p>
   </body>
</html>
```

Этот способ хорошо подходит для замены символов, не отображаемых в некоторых броузерах. Внешний вид документа HTML показан на рис. А.14.

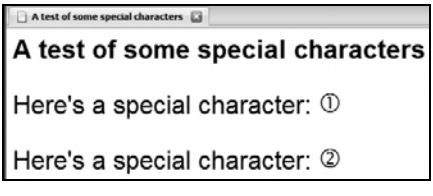


Рис. А.14. *Документ HTML, сгенерированный с помощью карты символов*

<xsl:param>

Определяет имя и значение параметра, используемого в таблице стилей. Элемент располагается на верхнем уровне или внутри элемента `<xsl:template>`. Если `<xsl:param>` располагается на верхнем уровне, то он описывает глобальный параметр, видимый в любой точке таблицы стилей. Значение параметра по умолчанию определяется одним из двух способов: либо в атрибуте `select`, либо в самом элементе `<xsl:param>`. *[2.0]* В XSLT 2.0 элемент `<xsl:param>` также может находиться в новом элементе `<xsl:function>`.

Категория

Инструкция.

Обязательный атрибут

`name`

Определяет имя параметра.

Необязательные атрибуты

`select`

Содержит выражение XPath, определяющее значение параметра.

[2.0 – Schema] `as`

Определяет тип данных параметра. Им может быть любой из встроенных типов данных или же тип данных, определенный в схеме, если вы используете схемосовместимый процессор XSLT 2.0. Например, `as="xs:integer"` задает параметр, который является целым числом, а `as="xs:string*"` – параметр, который является последовательностью из нуля или более строк. При использовании схемосовместимого процессора XSLT запись `as="schema-element(po:purchase-order)"` означает, что параметр должен быть экземпляром `po:purchase-order`, прошедшим проверку на действительность.

[2.0] `required`

Определяет, является ли параметр обязательным. Допустимые значения: `yes` и `no`. Если параметр принадлежит `<xsl:function>`, этот атрибут не должен определяться. Для параметров шаблонов и таблиц стилей по умолчанию параметр не является обязательным (`required="no"`).

Наличие атрибута `required` одновременно со значением по умолчанию у `<xsl:param>` является статической ошибкой. Обязательные параметры не могут иметь атрибута `select`, и они должны быть пустыми.

Использование атрибута `required` для параметра, определяемого в элементе `<xsl:function>`, также является статической ошибкой.

[2.0] `tunnel`

Определяет, является ли параметр туннельным. Допустимые значения: `yes` и `no`; по умолчанию используется значение `no`. (Туннельные параметры подробно рассматриваются в разделе «Туннельные параметры» главы 5.)

Содержимое

Если атрибут `select` используется, элемент `<xsl:param>` должен быть пустым. В противном случае он содержит шаблон XSLT.

[2.0] В XSLT 2.0 это правило соблюдается более жестко. Наличие у непустого элемента `<xsl:param>` атрибута `select` является статической ошибкой.

Местонахождение

`<xsl:stylesheet>` и `<xsl:template>`. Если элемент `<xsl:param>` является дочерним элементом `<xsl:stylesheet>`, то он определяет глобальный параметр, видимый в любой точке таблицы стилей. XSLT не определяет механизм передачи глобальных параметров процессору XSLT; за информацией о нем обращайтесь к документации своего процессора XSLT. (Общий обзор механизма передачи параметров в самых распространенных процессорах XSLT приведен в разделе «Глобальные параметры» главы 5.)

[2.0] В XSLT 2.0 элемент `<xsl:param>` также может находиться в новом элементе `<xsl:function>`.

Где определяется

[1.0] XSLT, раздел 11 «Variables and Parameters».

[2.0] XSLT, раздел 9 «Variables and Parameters».

Пример

Следующая таблица стилей определяет несколько элементов <xsl:param>, глобальных и локальных. Обратите внимание: один из параметров представляет собой набор узлов; как упоминалось ранее, параметры могут относиться к любому из типов данных XPath или XSLT.

```
<?xml version="1.0"?>
<!-- param.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:param name="favoriteNumber" select="23"/>
  <xsl:param name="favoriteColor"/>

  <xsl:template match="/">
    <xsl:text>&#xA;</xsl:text>
    <xsl:value-of select="list/title"/>
    <xsl:text>&#xA;</xsl:text>
    <xsl:variable name="listitems" select="list/listitem"/>
    <xsl:call-template name="processListItems">
      <xsl:with-param name="items" select="$listitems"/>
      <xsl:with-param name="color" select="'yellow'"/>
      <xsl:with-param name="number" select="$favoriteNumber"/>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="processListItems">
    <xsl:param name="items"/>
    <xsl:param name="color" select="'blue'"/>

    <xsl:for-each select="$items">
      <xsl:value-of select="position()"/>
      <xsl:text>.  </xsl:text>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>

    <xsl:text>&#xA;</xsl:text>

    <xsl:text>Your favorite color is </xsl:text>
    <xsl:value-of select="$favoriteColor"/>
    <xsl:text>.</xsl:text>
    <xsl:text>&#xA;</xsl:text>
    <xsl:text>The color passed to this template is </xsl:text>
    <xsl:value-of select="$color"/>
    <xsl:text>.</xsl:text>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей используется для преобразования следующего документа:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list>
  <title>A few of my favorite albums</title>
```

```
  <listitem>A Love Supreme</listitem>
  <listitem>Beat Crazy</listitem>
  <listitem>You Could Have it So Much Better</listitem>
  <listitem>Kind of Blue</listitem>
  <listitem>London Calling</listitem>
  <listitem>Remain in Light</listitem>
  <listitem>The Joshua Tree</listitem>
  <listitem>The Indestructible Beat of Soweto</listitem>
</list>
```

Результат:

```
A few of my favorite albums
1.  A Love Supreme
2.  Beat Crazy
3.  You Could Have it So Much Better
4.  Kind of Blue
5.  London Calling
6.  Remain in Light
7.  The Joshua Tree
8.  The Indestructible Beat of Soweto

Your favorite color is purple.
The color passed to this template is yellow.
```

Обратите внимание: при вызове шаблона `processListItems` передаются три параметра, только два из которых определяются внутри шаблона. В XSLT 1.0 неопределенные параметры просто игнорируются; в XSLT 2.0 они приводят к фатальной ошибке. Если преобразовать таблицу стилей к виду `<xsl:stylesheet version="2.0" ...`, она не будет работать. Обратите также внимание на то, что параметр `$items` является набором узлов, поэтому мы можем использовать его в элементе `<xsl:for-each>`.

Для получения приведенных ранее результатов мы передаем значение `purple` процессору XSLT. При использовании Saxon командная строка выглядит так:

```
java net.sf.saxon.Transform albums.xml param.xsl favoriteColor=purple
```

Процессору Xalan-J значение передается следующим образом:

```
java org.apache.xalan.xslt.Process -in albums.xml -xsl param.xsl
  -param favoriteColor purple
```

(Команда должна размещаться в одной строке.) За подробной информацией о глобальных параметрах и передаче их в разных процессорах XSLT обращайтесь к разделу «Глобальные параметры» главы 5.

В качестве дополнительного примера рассмотрим таблицу стилей XSLT 2.0 с использованием обязательных параметров, включая обязательный глобальный параметр, который должен задаваться при активизации таблицы стилей:

```
<?xml version="1.0"?>
<!-- param2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:param name="favoriteNumber"
    required="yes" as="xs:integer"/>
```

```
  <xsl:template match="/">
    <xsl:text>&#xA;</xsl:text>
    <xsl:value-of select="list/title"/>
    <xsl:text>&#xA;</xsl:text>
    <xsl:variable name="listitems" select="list/listitem"/>
    <xsl:call-template name="processListItems">
      <xsl:with-param name="items" select="$listitems"/>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="processListItems">
    <xsl:param name="items" required="yes"/>

    <xsl:for-each select="$items">
      <xsl:value-of select="position()"/>
      <xsl:text>.  </xsl:text>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>

    <xsl:text>&#xA;</xsl:text>

    <xsl:text>Your favorite number is </xsl:text>
    <xsl:value-of select="$favoriteNumber"/>
    <xsl:text>.</xsl:text>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Эта таблица стилей выдает практически такой же результат, что и предыдущая, но заставляет передавать при вызове обязательный параметр. Мы рассмотрим четыре варианта вызова таблицы стилей в Saxon. В первом варианте обязательный параметр не передается:

```
C:\>java net.sf.saxon.Transform albums.xml param2.xsl
Error
  XTDE0050: No value supplied for required parameter favoriteNumber
Transformation failed: Run-time errors were reported
```

Во втором варианте значение обязательного параметра передается, но не является целым числом:

```
C:\>java net.sf.saxon.Transform albums.xml param2.xsl favoriteNumber=yellow
Validation error
  FORG0001: Cannot convert string "yellow" to an integer
Transformation failed: Run-time errors were reported
```

В третьем варианте передается допустимое число, но имя переменной вводится с ошибкой. В именах переменных в XSLT учитывается регистр символов, поэтому `favoritenumber` и `favoriteNumber` — две разные переменные.

```
C:\>java net.sf.saxon.Transform albums.xml param2.xsl favoritenumber=23
Error
  XTDE0050: No value supplied for required parameter favoriteNumber
Transformation failed: Run-time errors were reported
```

Наконец, при последнем вызове передается обязательный параметр с допустимым значением, и на этот раз все работает нормально:

```
C:\>java net.sf.saxon.Transform albums.xml param2.xsl
favoriteNumber=23

Albums I've bought recently:
1.  The Sacred Art of Dub
2.  Only the Poor Man Feel It
3.  Excitable Boy
4.  Aki Special
5.  Combat Rock
6.  Talking Timbuktu
7.  The Birth of the Cool

Your favorite number is 23.
```

[2.0] <xsl:perform-sort>

Сортирует последовательность. Сортируемая последовательность определяется атрибутом `select` либо строится внутри самого элемента `<xsl:perform-sort>`.

Категория

Инструкция.

Обязательные атрибуты

Нет.

Необязательный атрибут

`select`

Выражение XPath, определяющее сортируемые объекты. При наличии атрибута `select` элемент `<xsl:perform-sort>` может содержать только элементы `<xsl:sort>` и `<xsl:fallback>`. Иначе говоря, с атрибутом `select` элементу `<xsl:perform-sort>` не разрешается строить последовательность, потому что она уже задана атрибутом `select`.

Содержимое

Элемент `<xsl:perform-sort>` содержит один или несколько элементов `<xsl:sort>`, а также произвольное количество элементов `<xsl:fallback>`. Если атрибут `select` не определен, `<xsl:perform-sort>` может содержать конструктор последовательности. Если атрибут `select` отсутствует, а элемент `<xsl:perform-sort>` не создает последовательность, то результатом является пустая последовательность.

Местонахождение

Любой элемент XSLT, модель содержимого которого представляет собой конструктор последовательности или элемент с результатом-литералом.

Где определяется

XSLT, раздел 13.2 «Creating a Sorted Sequence».

Пример

Начнем с элемента `<xsl:perform-sort>`, использующего атрибут `select` для создания отсортированной последовательности. Таблица стилей:

```
<?xml version="1.0"?>
<!-- perform-sort4.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:variable name="vendorsInOrder" as="xs:string*">
    <xsl:perform-sort select="/report/brand/name">
      <xsl:sort select="."/>
    </xsl:perform-sort>
  </xsl:variable>

  <xsl:template match="/">
    <xsl:text>We sell these brands of chocolate:&#xA;&#xA;</xsl:text>
    <xsl:value-of select="$vendorsInOrder" separator="&#xA;"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей обрабатывает документ с данными о продажах шоколада:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Результат:

```
We sell these brands of chocolate:

Callebaut
Ghirardelli
Lindt
Perugina
Valrhona
```

Атрибут select элемента <xsl:perform-sort> создает последовательность всех элементов <name> в исходном документе. Имена элементов сортируются и возвращаются в виде последовательности, которая сохраняется в переменной vendorsInOrder. Учтите, что <xsl:perform-sort> возвращает последовательность из нуля, одного или нескольких объектов.

Элементы <xsl:sort> могут иметь произвольную сложность. В нашем примере создаваемая последовательность базируется на названиях брендов, но порядок сортировки определяется объемами продаж:

```
<?xml version="1.0"?>
<!-- perform-sort5.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:variable name="vendorsInOrder" as="xs:string*">
    <xsl:perform-sort select="/report/brand/name">
      <xsl:sort select="../units"
        data-type="number" order="descending"/>
    </xsl:perform-sort>
  </xsl:variable>

  <xsl:template match="/">
    <xsl:text>Here are our best-selling brands:&#xA;&#xA;</xsl:text>
    <xsl:value-of select="$vendorsInOrder" separator="&#xA;"/>
  </xsl:template>

</xsl:stylesheet>
```

Теперь результат выглядит иначе:

```
Here are our best-selling brands:

Lindt
Valrhona
Ghirardelli
Perugina
Callebaut
```

Для определения объектов последовательности по-прежнему используется атрибут select, но сортировка осуществляется по более сложному ключу. Чтобы привести пример элемента <xsl:perform-sort>, не использующего атрибут select, мы воспользуемся элементом <xsl:sequence> для выбора узлов:

```
<?xml version="1.0"?>
<!-- perform-sort6.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:variable name="vendorsInOrder" as="xs:string*">
    <xsl:perform-sort>
      <xsl:sort select="../units"
        data-type="number" order="descending"/>
```

```
      <xsl:sequence select="/report/brand/name"/>
    </xsl:perform-sort>
  </xsl:variable>

  <xsl:template match="/">
    <xsl:text>Here are our best-selling brands:&#xA;&#xA;</xsl:text>
    <xsl:value-of select="$vendorsInOrder" separator="&#xA;"/>
  </xsl:template>

</xsl:stylesheet>
```

Мы получаем прежний результат, но с более странным синтаксисом. В элементе `<xsl:perform-sort>` на первом месте должны размещаться элементы `<xsl:sort>` (один или несколько). Они определяют ключ сортировки для последовательности объектов, создаваемой в самом элементе `<xsl:perform-sort>`. При желании можно сгенерировать и более сложную последовательность:

```
<?xml version="1.0"?>
<!-- perform-sort7.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:variable name="vendorsInOrder" as="xs:string*">
    <xsl:perform-sort>
      <xsl:sort select="."
        data-type="number" order="descending"/>
      <xsl:sequence select="/report/brand/units"/>
      <xsl:sequence select="('3829', '28852', '18831')"/>
    </xsl:perform-sort>
  </xsl:variable>

  <xsl:template match="/">
    <xsl:text>Here are our sales figures:&#xA;&#xA;</xsl:text>
    <xsl:value-of select="$vendorsInOrder" separator="&#xA;"/>
  </xsl:template>

</xsl:stylesheet>
```

В таблицу включены фиктивные данные о продажах. Результат показывает, что дополнительные значения были включены в последовательность до сортировки:

```
Here are our sales figures:

28852
27408
22101
19268
18831
14336
8203
3829
```

Данные сортируются по убыванию, потому что в элемент `<xsl:sort>` включен атрибут `order="descending"`.

<xsl:preserve-space>

Определяет элементы исходного документа, в которых должны быть сохранены пропуски.

Категория

Элемент верхнего уровня.

Обязательный атрибут

`elements`

Содержит разделенный пробелами список элементов исходного документа, в которых должны быть сохранены незначащие пропуски. *Незначащими пропусками* обычно называются текстовые узлы, не содержащие ничего, кроме пропусков; пропуски в тексте и вокруг него сохраняются всегда. Атрибут `elements` может также содержать значение `*`, которое означает, что пропуски должны сохраняться во всех элементах, не указанных в элементе `<xsl:strip-space>`. (Запись `<xsl:preserve-space elements="*"/>` допустима, однако вам не придется использовать ее. XSLT по умолчанию сохраняет текстовые узлы, состоящие из одних пропусков.) Допускается также использование записи вида `elements="po:*"` для обозначения всех элементов в пространстве имен `po`.

[2.0] В XSLT 2.0 атрибут `elements` допускает использование метасимволов в префиксе пространства имен. Например, запись `elements="*:title"` обозначает все элементы `<title>` независимо от их пространства имен. Как и в XSLT 1.0, значение `elements="dc:*"` описывает все элементы пространства имен `dc`.

В спецификации XML определяется редко используемый атрибут `xml:space`. Атрибут `xml:space` может принимать значения `preserve` или `default`. Если к некоторому элементу исходного документа XML применяется атрибут `xml:space="preserve"`, все пропуски сохраняются независимо от элементов `<xsl:preserve-space>` и `<xsl:strip-space>`.

Необязательные атрибуты

Нет.

Содержимое

Нет. `<xsl:preserve-space>` является пустым элементом.

Местонахождение

`<xsl:preserve-space>` относится к элементам верхнего уровня, а его родителем может быть только элемент `<xsl:stylesheet>`.

Где определяется

[1.0] XSLT, раздел 3.4 «Whitespace Stripping».

[2.0] XSLT section 4.4 «Stripping Whitespace from a Source Tree».

Пример

Для демонстрации применения `<preserve-space>` используется следующая таблица стилей:

```
<?xml version="1.0"?>
<!-- preserve-space.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:strip-space elements="*"/>
  <xsl:preserve-space elements="listing"/>

  <xsl:template match="/">
    <xsl:text>&#xA;</xsl:text>
    <xsl:value-of select="/code-sample/title"/>
    <xsl:text>&#xA;</xsl:text>
    <xsl:for-each select="/code-sample/listing">
      <xsl:value-of select="."/>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей обрабатывает следующий документ XML:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- whitespace.xml -->
<code-sample>
  <title>Conditional variable initialization:</title>
  <listing>
    <type>int</type><var> y</var><endstmt>;</endstmt>
    <type>int</type><var> x</var><endstmt>;</endstmt>

    <var>y</var> <op>=</op> <const>23</const><endstmt>;</endstmt>

    <block>
      <keyword>if</keyword> (<var>y</var> <comp>></comp> <const>10</const>)
        <var>x</var> <op>=</op> <const>5</const><endstmt>;</endstmt>
      <keyword>else </keyword>
      <keyword>if</keyword> (<var>y</var> <comp>></comp> <const>5</const>)
        <var>x</var> <op>=</op> <const>3</const><endstmt>;</endstmt>
      <keyword>else </keyword>
        <var>x</var> <op>=</op> <const>1</const><endstmt>;</endstmt>
    </block>
  </listing>
</code-sample>
```

При обработке документа будет получен следующий результат:

```
Conditional variable initialization:

    int y;
    int x;

    y = 23;

    if (y>10)
        x=5;else if (y>5)
        x=3;else x=1;
```

Элемент <xsl:strip-space> удаляет пропуски их всех элементов в документе, поэтому мы используем <xsl:preserve-space> для сохранения пропусков в элементе <listing>. Обратите внимание: все пропуски в элементе <block> были удалены. Элемент

<xsl:preserve-space> удобен в тех ситуациях, когда требуется исключить отдельный элемент из общего удаления пропусков; по умолчанию пропуски из узлов не удаляются. Сравните этот пример с приведенным в описании элемента <strip-space>.

<xsl:processing-instruction>

Создает инструкцию по обработке в выходном документе.

Категория

Инструкция.

Обязательный атрибут

name

Определяет имя инструкции по обработке.

Необязательный атрибут

[2.0] select

Определяет выражение, создающее данные для инструкции по обработке. Если атрибут select используется, элемент <xsl:processing-instruction> должен быть пустым.

Содержимое

Шаблон XSLT. Содержимое шаблона образует данные для инструкции по обработке.

Местонахождение

Элемент <xsl:processing-instruction> находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 7.3 «Creating Processing Instructions».

[2.0] XSLT, раздел 11.6 «Creating Processing Instructions».

Пример

Следующая таблица стилей включает в документ XML инструкцию по обработке, связывающую этот документ XML с таблицей стилей *docbook.xsl*. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- processing-instruction.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml"/>

  <xsl:template match="/">
    <xsl:processing-instruction name="xml-stylesheet">href="docbook/
html/docbook.xsl" type="text/xsl"</xsl:processing-instruction>
    <xsl:copy-of select="."/>
  </xsl:template>

</xsl:stylesheet>
```

(Элемент <xsl:processing-instruction> должен размещаться в одной строке.) Таблица стилей просто копирует исходный документ в дерево результата при помощи элемента <xsl:copy-of>, добавляя к нему инструкцию по обработке. Таблица стилей используется со следующим документом XML:

```
<?xml version="1.0"?>
<!-- greeting.xml -->
<greeting>
  Hello, World!
</greeting>
```

При выполнении преобразования будет получен следующий результат:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="docbook/html/docbook.xsl" type="text/xsl"?>
<greeting>
  Hello, World!
</greeting>
```

Обратите внимание: содержимое инструкции по обработке представляет собой текст. Сгенерированная инструкция выглядит так, словно она содержит два атрибута, однако попытка создать инструкцию с атрибутами завершается неудачей:

```
<?xml version="1.0"?>
<!-- processing-instruction-doesnt-work.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml"/>

  <xsl:template match="/">
    <xsl:processing-instruction name="xml-stylesheet">

      <!-- Не работает!  Элемент <xsl:processing-instruction> не может содержать
           элементы <xsl:attribute>. -->

      <xsl:attribute name="href">
        <xsl:text>docbook/html/docbook.xsl</xsl:text>
      </xsl:attribute>
      <xsl:attribute name="type">
        <xsl:text>text/xsl</xsl:text>
      </xsl:attribute>
    </xsl:processing-instruction>
    <xsl:copy-of select="."/>
  </xsl:template>

</xsl:stylesheet>
```

Если вы попытаетесь поступить подобным образом, Xalan выдаст исключение. Saxon исключения не выдает, но и действительную инструкцию по обработке не генерирует.

[2.0] <xsl:result-document>

Создает дерево результата. Как правило, дерево результата записывается в файл, хотя процессор XSLT 2.0 этого делать не обязан. Инструкция <xsl:result-document> удобна для построения нескольких файлов по одной таблице стилей. Кроме того, она позволяет сгенерировать имя выходного файла на стадии выполнения.

Категория

Инструкция.

Обязательные атрибуты

Нет.

Необязательные атрибуты

`format`

Ссылается на спецификацию вывода, определяемую для именованного элемента `<xsl:output>`. Вы можете определить набор свойств (метод вывода, кодировку символов и т. д.), а затем многократно использовать готовую спецификацию, указывая ее атрибут `name` в соответствующем элементе `<xsl:output>`.

`href`

Определяет URI-адрес итогового документа. Обычно используется для определения имени файла итогового документа, хотя процессор XSLT может использовать этот URI-адрес любым способом на свое усмотрение.

[2.0 – Schema] `type`

Определяет тип данных элемента документа. Им может быть любой из встроенных типов данных или же тип данных, определенный в схеме (при использовании схемосовместимого процессора XSLT 2.0).

Атрибуты `type` и `validation` являются взаимоисключающими.

[2.0 – Schema] `validation`

Определяет способ проверки нового элемента документа. Атрибут `validation` имеет четыре допустимых значения: `strict`, `lax`, `preserve` и `strip`.

Атрибут `validation="strict"` означает, что процессор XSLT должен искать во всех объявленных схемах объявление элемента (`<xs:element>`) с таким же именем, как у элемента документа. Если процессору не удается найти подходящее объявление, происходит фатальная ошибка. Если же процессор находит объявление элемента документа, он проверяет по нему новый документ.

Значение `validation="lax"` работает аналогично `validation="strict"`, но если процессору не удается найти объявление элемента документа ни в одной из объявленных схем, ошибка не происходит. В этом случае элемент документа имеет обозначение типа `xs:untyped`.

Со значением `validation="preserve"` созданный элемент получает обозначение типа `xs:anyType`, а обозначения типов всех узлов, содержащихся в новом элементе документа, сохраняются без изменений. Проверка по схеме не выполняется.

Наконец, значение `validation="strip"` создает элемент документа с типом `xs:anyType`.

Атрибуты `validation` и `type` являются взаимоисключающими.

`method`

Определяет метод вывода дерева результата. Допустимые значения: `xml`, `html`, `xhtml` и `text`. Процессор XSLT может дополнить список другими значениями; за информацией о поддержке других методов вывода обращайтесь к документации своего процессора.

byte-order-mark

Определяет, должна ли метка порядка байтов записываться в выходной документ. Допустимые значения: yes и no. По умолчанию значение yes используется для кодировки UTF-16, тогда как значение по умолчанию для UTF-8 зависит от процессора XSLT. Для всех остальных кодировок по умолчанию используется значение no.

cdata-section-elements

Перечисляет элементы, которые записываются в выходной документ в виде секций CDATA. Соблюдение всех ограничений и правил экранирования для секций CDATA обеспечивается процессором XSLT. Если потребуется перечислить более одного элемента, разделите их имена одним или несколькими символами-пропусками. Атрибут используется только в режимах method="xml" и method="xhtml".

doctype-public

Определяет значение атрибута PUBLIC в объявлении DOCTYPE выходного документа. Атрибут определяет открытый идентификатор DTD выходного документа. Игнорируется при наличии атрибута method="text".

doctype-system

Определяет значение атрибута SYSTEM в объявлении DOCTYPE в выходном документе. Атрибут определяет системный идентификатор DTD выходного документа. Игнорируется при наличии атрибута method="text".

encoding

Определяет значение атрибута encoding в объявлении XML выходного документа. Игнорируется при наличии атрибута method="text".

escape-uri-attributes

Определяет, должны ли специальные символы в атрибутах HTML и XHTML с URI-значениями заменяться своими шестнадцатеричными эквивалентами. Например, если URI-адрес содержит пробелы, в преобразованном значении каждый пробел заменяется обозначением %20.

include-content-type

Определяет, должен ли тип содержимого записываться в выходной документ. Допустимые значения: yes и no; по умолчанию используется значение yes. Например, при использовании атрибута include-content-type для method="html" в элемент <head> документа HTML включается элемент <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">.

indent

Определяет наличие отступов в тегах выходного документа. Допустимые значения: yes и no. Атрибут игнорируется в режиме method="text".

media-type

Определяет тип MIME для выходного документа. По умолчанию используются значения text/xml для method="xml", text/html для method="html" и method="xhtml" и text/plain для method="text".

normalization-form

Допустимые значения: NFC, NFD, NFKC, NFKD, fully-normalized и none. Процессоры XSLT могут поддерживать и другие значения; за информацией о поддержке других форм нормализации обращайтесь к документации своего процессора.

`omit-xml-declaration`

Определяет, должно ли объявление XML быть опущено в выходном документе. Допустимые значения: `yes` и `no`; по умолчанию используется `no`.

`standalone`

Определяет, должно ли объявление XML включать атрибут `standalone`. Допустимые значения: `yes` (объявление содержит атрибут `standalone="yes"`), `no` (объявление содержит атрибут `standalone="no"`) и (*[2.0]*) `omit` (объявление не содержит атрибут `standalone`).

`undeclare-prefixes`

Определяет, должен ли выходной документ включать отменяющие объявления пространств имен (такие объявления связывают префикс пространства имен с пустой строкой, например `xmlns:doug=""`). Допустимые значения: `yes` и `no`. За полной информацией об этом малопонятном атрибуте обращайтесь к разделу 20 спецификации XSLT 2.0.

`use-character-maps`

Определяет список разделенных пробелами именованных карт символов, используемых при построении выходного документа.

`output-version`

Переопределяет атрибут `version` элемента `<xsl:output>`.

Содержимое

Конструктор последовательности.

Местонахождение

Элемент `<xsl:template>`.

Где определяется

XSLT, раздел 19.1 «Creating Final Result Trees».

Пример

Для демонстрации возможностей `<xsl:result-document>` используется следующий небольшой документ DocBook:

```
<?xml version="1.0"?>
<!-- chapters.xml -->
<book>
  <title>XSLT Topics</title>
  <chapter>
    <title>XPath</title>
    <para>If this chapter had any text, it would appear here.</para>
  </chapter>
  <chapter>
    <title>Stylesheet Basics</title>
    <para>If this chapter had any text, it would appear here.</para>
  </chapter>
  <chapter>
    <title>Branching and Control Elements</title>
    <para>If this chapter had any text, it would appear here.</para>
  </chapter>
```

```
  <chapter>
    <title>Functions</title>
    <para>If this chapter had any text, it would appear here.</para>
  </chapter>
  <chapter>
    <title>Creating Links and Cross-References</title>
    <para>If this chapter had any text, it would appear here.</para>
  </chapter>
  <chapter>
    <title>Sorting and Grouping Elements</title>
    <para>If this chapter had any text, it would appear here.</para>
  </chapter>
  <chapter>
    <title>Combining XML Documents</title>
    <para>If this chapter had any text, it would appear here.</para>
  </chapter>
</book>
```

Таблица стилей используется для создания нескольких документов. Сначала создается общий файл HTML с перечислением всех названий (<title>**) элементов** <chapter>**, определенных в документе. Каждое название оформляется в виде ссылки на отдельный файл HTML, создаваемый при помощи** <xsl:result-document>**. Элемент** <xsl:result-document> **создает отдельный файл HTML для каждого элемента** <chapter>**. Таблица стилей выглядит так:**

```
<?xml version="1.0" encoding="utf-8"?>
<!-- result-document.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" include-content-type="no"/>

  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of select="book/title"/></title>
      </head>
      <body>
        <h1><xsl:value-of select="book/title"/></h1>
        <p>Here are some interesting XSLT topics:</p>
        <ul>
          <xsl:for-each select="book/chapter">
            <li>
              <a href="{concat('chapter', position(), '.html')}">
                <xsl:value-of select="title"/>
              </a>
            </li>
          </xsl:for-each>
        </ul>
        <xsl:for-each select="book/chapter">
          <xsl:result-document method="html"
            include-content-type="no"
            href="{concat('chapter', position(), '.html')}">
            <html>
              <head>
                <title><xsl:value-of select="title"/></title>
              </head>
              <body>
```

```
            <h1><xsl:value-of select="title"/></h1>
            <xsl:apply-templates select="*[position() > 1]"/>
          </body>
        </html>
      </xsl:result-document>
    </xsl:for-each>
  </body>
 </html>
</xsl:template>

<xsl:template match="para">
  <p><xsl:apply-templates select="*|text()"/></p>
</xsl:template>

</xsl:stylesheet>
```

Таблица стилей начинается с создания главного документа HTML. Элемент `<xsl:for-each select="book/chapter">` создает пункт списка и ссылку для каждого элемента `<chapter>` в исходном документе. Затем другой элемент `<xsl:for-each>` при помощи `<xsl:result-document>` создает новый документ для каждого элемента `<chapter>`.

Обратите внимание на использование шаблона значения атрибута `{concat('chapter', position(), '.html')}` для создания имен файлов и ссылок. Можно было потребовать, чтобы в каждый элемент `<chapter>` включался специальный идентификационный атрибут, но проще воспользоваться жестко заданной схемой построения имен для создания файлов и ссылок. Четвертый элемент `<chapter>` соответствует файлу *chapter4.html*. Главный файл HTML выглядит так:

```
<html>
   <head>
      <title>XSLT Topics</title>
   </head>
   <body>
      <h1>XSLT Topics</h1>
      <p>Here are some interesting XSLT topics:</p>
      <ul>
         <li><a href="chapter1.html">XPath</a></li>
         <li><a href="chapter2.html">Stylesheet Basics</a></li>
         <li><a href="chapter3.html">Branching and Control Elements</a></li>
         <li><a href="chapter4.html">Functions</a></li>
         <li><a href="chapter5.html">Creating Links and Cross-References</a></li>
         <li><a href="chapter6.html">Sorting and Grouping Elements</a></li>
         <li><a href="chapter7.html">Combining XML Documents</a></li>
      </ul>
   </body>
</html>
```

Каждый файл HTML отдельной главы выглядит примерно так:

```
<html>
   <head>
      <title>XPath</title>
   </head>
   <body>
      <h1>XPath</h1>
      <p>If this chapter had any text, it would appear here.</p>
   </body>
</html>
```

На рис. A.15 показано, как главный файл HTML выглядит в броузере.

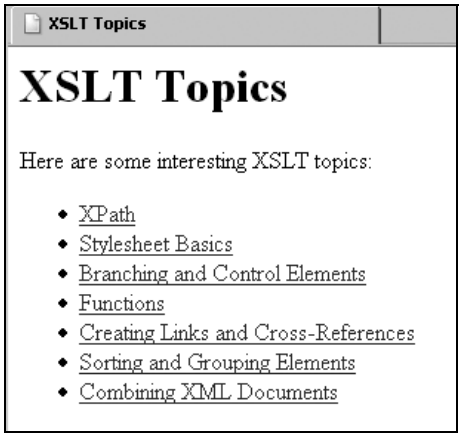


***Рис. A.15.** Файл HTML со ссылками на документы, созданные при помощи элемента <xsl:result-document>*

Щелчок на любой ссылке открывает один документ отдельной главы, созданный элементом `<xsl:result-document>`; внешний вид этого документа показан на рис. A.16.

***Рис. A.16.** Один из документов, созданных при помощи элемента <xsl:result-document>*

Элемент `<xsl:result-document>` официально заменяет стандартные элементы расширения в большинстве процессоров XSLT 1.0. Для создания нескольких выходных файлов в процессорах XSLT 1.0 приходилось вызывать элементы расширения, поддерживаемые конкретным процессором. Многие стандартные механизмы XSLT 1.0 подробно рассматриваются в разделе «Функция document() и сортировка» главы 9.

[2.0] <xsl:sequence>

Создает последовательность узлов и/или атомарных значений.

Категория

Инструкция.

Обязательный атрибут

`select`

Выражение XPath, определяющее содержимое последовательности.

Необязательные атрибуты

Нет.

Содержимое

Нуль или более элементов `<xsl:fallback>`. Элемент `<xsl:fallback>` определяет действия, которые должны быть выполнены при обнаружении элемента, не поддерживаемого процессором XSLT. Элементы `<xsl:fallback>` чаще всего задаются для процессоров XSLT, работающих в режиме прямой совместимости.

Местонахождение

Шаблон.

Где определяется

XSLT, раздел 11.10 «Constructing Sequences».

Пример

Далее приводится простой пример с созданием последовательности в элементе `<xsl:variable>`. Таблица стилей выглядит так:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- sequence.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="sales" as="xs:integer*">
      <xsl:for-each select="/report/brand/units">
        <xsl:if test=". > 10000">
          <xsl:sequence select="."/>
        </xsl:if>
      </xsl:for-each>
    </xsl:variable>

    <xsl:value-of select="/report/title"/>
    <xsl:text>&#xA;&#xA;Sales figures:  </xsl:text>
    <xsl:value-of select="$sales" separator=", "/>
    <xsl:text>&#xA;&#xA;Sequence total:   &#x9;</xsl:text>
    <xsl:value-of select="format-number(sum($sales), '$#,###.00')"/>
    <xsl:text>&#xA;Sequence average:&#x9;</xsl:text>
    <xsl:value-of select="format-number(avg($sales), '$#,###.00')"/>
  </xsl:template>

</xsl:stylesheet>
```

В цикле `<xsl:for-each>` каждый элемент `<xsl:sequence>` создает последовательность из одного узла. Значение переменной представляет собой последовательность значений

xs:integer, созданных на базе всех объектов (узлов и атомов), сгенерированных элементами <xsl:sequence>. (Иначе говоря, все, что выбирается элементом <xsl:sequence>, преобразуется в xs:integer, а затем становится частью переменной.) Применим таблицу стилей к отчету с данными о продажах шоколада:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Наша таблица стилей создает последовательность из всех элементов <units>, превышающих 10000. Результат выглядит так:

```
Chocolate bar sales

Sales figures:  27408, 22101, 14336, 19268

Sequence total:          $83,113.00
Sequence average:        $20,778.25
```

Помните, что последовательность может содержать произвольную комбинацию узлов и атомарных значений. Если нам вдруг захочется повысить объемы продаж, в последовательность можно включить дополнительные значения:

```
<!-- sequence2.xsl -->
...
<xsl:variable name="sales" as="xs:integer*">
  <xsl:for-each select="/report/brand/units">
    <xsl:if test=". > 10000">
      <xsl:sequence select="."/>
    </xsl:if>
  </xsl:for-each>
  <xsl:sequence select="(80000, 75000, 65000)"/>
</xsl:variable>
...
```

С новыми данными отчет о продажах выглядит намного внушительнее:

```
Chocolate bar sales

Sales figures:  27408, 22101, 14336, 19268, 80000, 75000, 65000

Sequence total:          $303,113.00
Sequence average:        $43,301.86
```

Напомню, что переменная содержит последовательность типа `xs:integer*`, поэтому все выбранные узлы и атомарные значения преобразуются в `xs:integer`, прежде чем стать частью переменной.

<xsl:sort>

Определяет ключ сортировки для текущего контекста. Первый элемент `<xsl:sort>` определяет первичный ключ сортировки, второй элемент `<xsl:sort>` определяет вторичный ключ, и так далее. Количество элементов `<xsl:sort>` не ограничивается.

Категория

Вложенная инструкция.

Обязательные атрибуты

Нет.

Необязательные атрибуты

`select`

Определяет сортируемые узлы. Если атрибут `select` отсутствует, сортируемые узлы определяются содержимым элемента `<xsl:sort>`.

`lang`

Строка, определяющая язык, алфавит которого должен использоваться при сортировке. Коды языков определяются в документе RFC1766, доступном по адресу *http://www.ietf.org/rfc/rfc1766.txt*. Не все процессоры поддерживают все возможные языки; обращайтесь к документации своего процессора XSLT.

`data-type`

Атрибут, определяющий тип сортируемых объектов. Допустимые значения: `number` и `text`; по умолчанию используется значение `text`. Процессор XSLT может поддерживать и другие значения. При сортировке значений `32 10 120` с атрибутом `data-type="text"` будет получен результат `10 120 32`, тогда как при сортировке с атрибутом `data-type="number"` будет получен результат `10 32 120`.

[2.0] В XSLT 2.0 можно задать полное имя, представляющее другой тип данных. Способ обработки этого значения зависит от конкретного процессора XSLT.

`order`

Атрибут, определяющий порядок сортировки. Допустимые значения: `ascending` (по возрастанию) и `descending` (по убыванию).

`case-order`

Атрибут, определяющий относительный порядок сортировки букв верхнего и нижнего регистра. Допустимые значения: `upper-first` (начиная с верхнего регистра) и `lower-first` (начиная с нижнего регистра).

[2.0] collation

Упорядочение, используемое при сортировке. Например, в испанском языке символ *ll* должен следовать после двух букв *lz*, поэтому в английской схеме упорядочения испанские слова сортируются некорректно.

Значение атрибута представляет собой URI-адрес; способ использования URI-адреса для задания конкретной схемы зависит от процессора XSLT. Если процессор XSLT 2.0 не поддерживает запрашиваемую схему упорядочения, используется упорядочение по умолчанию.

[2.0] stable

Допустимые значения: yes и no; по умолчанию используется значение yes. Только первый элемент <xsl:sort> может обладать атрибутом stable. При использовании атрибута stable="yes" элементы с одинаковым значением ключа сортировки размещаются в порядке их следования в документе. Допустим, сортируются следующие элементы:

```
<statelist>
  <title>U.S. Road trips I've made recently</title>
  <state year="2006" month="4">California</state>
  <state year="2005" month="12">Texas</state>
  <state year="2006" month="2">Washington</state>
  <state year="2006" month="3">California</state>
</statelist>
```

Сортировка осуществляется по атрибуту year и значению элемента <state> (атрибут month игнорируется); у первого и последнего элемента значения ключа сортировки совпадают. С атрибутом stable="yes" две записи всегда будут размещаться в результатах в исходном порядке документа. Если задать атрибут stable="no", процессор XSLT не обязан сохранять исходный порядок узлов.

Содержимое

[1.0] Нет.

[2.0] В XSLT 2.0 содержимое элемента <xsl:sort> может использоваться для определения сортируемых узлов. Естественно, наличие атрибута select у непустого элемента <xsl:sort> является фатальной ошибкой. Если атрибут select отсутствует у пустого элемента, по умолчанию используется текущий контекстный узел, что эквивалентно select=".".

Местонахождение

Элементы <xsl:apply-templates> и <xsl:for-each>.

[2.0] В XSLT 2.0 элемент <xsl:sort> также может находиться в новых элементах <xsl:for-each-group> и <xsl:perform-sort>.

Где определяется

[1.0] XSLT, раздел 10 «Sorting».

[2.0] XSLT, раздел 13 «Sorting».

Пример

Таблица стилей для демонстрации применения <xsl:sort>:

```
<?xml version="1.0"?>
<!-- sort.xsl -->
```

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;</xsl:text>
    <xsl:call-template name="ascending-alpha-sort">
      <xsl:with-param name="items" select="/sample/textlist/listitem"/>
    </xsl:call-template>
    <xsl:call-template name="ascending-alpha-sort">
      <xsl:with-param name="items" select="/sample/numericlist/listitem"/>
    </xsl:call-template>
    <xsl:call-template name="ascending-numeric-sort">
      <xsl:with-param name="items" select="/sample/numericlist/listitem"/>
    </xsl:call-template>
    <xsl:call-template name="descending-alpha-sort">
      <xsl:with-param name="items" select="/sample/textlist/listitem"/>
    </xsl:call-template>
    <xsl:call-template name="category-sort">
      <xsl:with-param name="items" select="/sample/textlist/listitem"/>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="ascending-alpha-sort">
    <xsl:param name="items"/>
    <xsl:text>Ascending text sort:</xsl:text>
    <xsl:text>&#xA;</xsl:text>
    <xsl:for-each select="$items">
      <xsl:sort select="."/>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

  <xsl:template name="descending-alpha-sort">
    <xsl:param name="items"/>
    <xsl:text>Descending text sort:</xsl:text>
    <xsl:text>&#xA;</xsl:text>
    <xsl:for-each select="$items">
      <xsl:sort select="." order="descending"/>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

  <xsl:template name="ascending-numeric-sort">
    <xsl:param name="items"/>
    <xsl:text>Ascending numeric sort:</xsl:text>
    <xsl:text>&#xA;</xsl:text>
    <xsl:for-each select="$items">
      <xsl:sort select="." data-type="number"/>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>
```

```
  <xsl:template name="category-sort">
    <xsl:param name="items"/>
    <xsl:text>Ascending category sort:</xsl:text>
    <xsl:text>&#xA;</xsl:text>
    <xsl:for-each select="$items">
      <xsl:sort select="@category"/>
      <xsl:sort select="."/>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей определяет четыре шаблона, сортирующих элементы `<listitem>` **в разном порядке или с разными значениями** `data-type`**. Четвертый шаблон** `category-sort` **сортирует сначала по атрибуту, а затем по значению текущего объекта. Таблица стилей применяется к следующему документу:**

```
<?xml version="1.0"?>
<!-- items-to-sort.xml -->
<sample>
  <numericlist>
    <listitem>1</listitem>
    <listitem>3</listitem>
    <listitem>23</listitem>
    <listitem>120</listitem>
    <listitem>2</listitem>
  </numericlist>
  <textlist>
    <listitem category="number">3</listitem>
    <listitem category="fruit">apple</listitem>
    <listitem category="fruit">orange</listitem>
    <listitem category="foreign-fruit">dragonfruit</listitem>
    <listitem category="foreign-fruit">carambola</listitem>
  </textlist>
  <datelist>
    <listitem>2006-12-25</listitem>
    <listitem>1995-04-21</listitem>
    <listitem>1965-06-19</listitem>
    <listitem>2007-01-01</listitem>
  </datelist>
  <spanishlist>
    <listitem>campo</listitem>
    <listitem>luna</listitem>
    <listitem>ciudad</listitem>
    <listitem>llaves</listitem>
    <listitem>chihuahua</listitem>
    <listitem>arroz</listitem>
    <listitem>limonada</listitem>
  </spanishlist>
</sample>
```

Результат:

```
Ascending text sort:
3
```

```
apple
carambola
dragonfruit
orange

Ascending text sort:
1
120
2
23
3

Ascending numeric sort:
1
2
3
23
120

Descending text sort:
orange
dragonfruit
carambola
apple
3

Ascending category sort:
carambola
dragonfruit
apple
orange
3
```

С атрибутом `data-type="numeric"` данные сортируются в числовом порядке.

[2.0] Чтобы продемонстрировать новые возможности XSLT 2.0, мы отсортируем те же данные по другой таблице стилей. Один шаблон преобразует дочерние узлы `<date-list>` в значения `xs:date`, а затем сортирует эти значения. Другой шаблон использует пользовательскую схему упорядочения для сортировки дочерних узлов `<spanish-list>`. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- sort2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;</xsl:text>
    <xsl:variable name="date-sequence" as="xs:date*">
      <xsl:for-each select="/sample/datelist/listitem">
        <xsl:value-of select="xs:date(.)"/>
      </xsl:for-each>
    </xsl:variable>
    <xsl:call-template name="ascending-date-sort">
      <xsl:with-param name="items" select="$date-sequence"/>
    </xsl:call-template>
```

```
    <xsl:call-template name="spanish-alpha-sort">
      <xsl:with-param name="items" select="/sample/spanishlist/listitem"/>
    </xsl:call-template>
    <xsl:call-template name="ascending-alpha-sort">
      <xsl:with-param name="items" select="/sample/spanishlist/listitem"/>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="ascending-date-sort">
    <xsl:param name="items" as="xs:date*"/>
    <xsl:text>Ascending date sort:</xsl:text>
    <xsl:text>&#xA;</xsl:text>
    <xsl:for-each select="$items">
      <xsl:sort/>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

  <xsl:template name="ascending-alpha-sort">
    <xsl:param name="items"/>
    <xsl:text>Ascending text sort:</xsl:text>
    <xsl:text>&#xA;</xsl:text>
    <xsl:for-each select="$items">
      <xsl:sort/>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

  <xsl:template name="spanish-alpha-sort">
    <xsl:param name="items"/>
    <xsl:text>Spanish alpha sort:</xsl:text>
    <xsl:text>&#xA;</xsl:text>
    <xsl:for-each select="$items">
      <xsl:sort
        collation="{concat('http://saxon.sf.net/collation?',
                   'class=com.oreilly.xslt.SpanishCollation;')}"/>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Ascending date sort:
1965-06-19
1995-04-21
2006-12-25
2007-01-01

Spanish alpha sort:
arroz
campo
```

```
ciudad
chihuahua
limonada
luna
llaves

Ascending text sort:
arroz
campo
chihuahua
ciudad
limonada
llaves
luna
```

Даты сортируются как значения xs:date, хотя конечный результат оказывается таким же, как при текстовой сортировке. В качестве схемы упорядочения для испанского языка мы указываем класс Java, сравнивающий строки для определения нужного порядка. Такой способ задания класса Java относится к специфике Saxon (XSLT 2.0). Чтобы узнать, какие пользовательские схемы упорядочения поддерживает ваш процессор XSLT (и поддерживает ли их вообще), обратитесь к его документации.

Пользовательские схемы упорядочения рассматриваются в главе 9.

<xsl:strip-space>

Определяет элементы исходного документа, в которых должны быть удалены незначащие пропуски.

Категория

Элемент верхнего уровня.

Обязательный атрибут

elements

Содержит разделенный пробелами список имен элементов или текстовых узлов, в которых должны быть удалены незначащие пропуски. *Незначащими пропусками* обычно называются текстовые узлы, не содержащие ничего, кроме пропусков; пропуски в тексте и вокруг него сохраняются всегда. Атрибут elements может также содержать значение *, которое означает, что пропуски должны сохраняться во всех элементах, не указанных в элементе <xsl:preserve-space>. Допускается также использование записи вида elements="auth:*" для обозначения всех элементов в пространстве имен auth.

[2.0] В XSLT 2.0 атрибут elements допускает использование метасимволов в префиксе пространства имен. Например, запись *:title обозначает все элементы <title> независимо от их пространства имен.

В спецификации XML определяется редко используемый атрибут xml:space. Атрибут xml:space может принимать значения preserve или default. Если к некоторому элементу исходного документа XML применяется атрибут xml:space="preserve", все пропуски сохраняются независимо от элементов <xsl:preserve-space> и <xsl:strip-space>.

Необязательные атрибуты

Нет.

Содержимое

Нет. <xsl:strip-space> является пустым элементом.

Местонахождение

<xsl:strip-space> относится к элементам верхнего уровня, а его родителем может быть только элемент <xsl:stylesheet>.

Где определяется

[1.0] XSLT, раздел 3.4 «Whitespace Stripping».

[2.0] XSLT section 4.4 «Stripping Whitespace from a Source Tree».

Пример

Для демонстрации применения <xsl:strip-space> используется следующая таблица стилей:

```
<?xml version="1.0"?>
<!-- strip-space.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:strip-space elements="*"/>

  <xsl:template match="/">
    <xsl:text>&#xA;</xsl:text>
    <xsl:value-of select="/code-sample/title"/>
    <xsl:text>&#xA;</xsl:text>
    <xsl:for-each select="/code-sample/listing">
      <xsl:value-of select="."/>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей используется для обработки следующего документа:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- whitespace.xml -->
<code-sample>
  <title>Conditional variable initialization:</title>
  <listing>
    <type>int</type><var> y</var><endstmt>;</endstmt>
    <type>int</type><var> x</var><endstmt>;</endstmt>

    <var>y</var> <op>=</op> <const>23</const><endstmt>;</endstmt>

    <block>
      <keyword>if</keyword> (<var>y</var> <comp>></comp> <const>10</const>)
        <var>x</var> <op>=</op> <const>5</const><endstmt>;</endstmt>
      <keyword>else </keyword>
      <keyword>if</keyword> (<var>y</var> <comp>></comp> <const>5</const>)
        <var>x</var> <op>=</op> <const>3</const><endstmt>;</endstmt>
```

```
        <keyword>else </keyword>
          <var>x</var> <op>=</op> <const>1</const><endstmt>;</endstmt>
      </block>
    </listing>
  </code-sample>
```

Результат:

```
Conditional variable initialization:
int y;int x;y=23;if (y>10)
        x=5;else if (y>5)
        x=3;else x=1;
```

Обратите внимание: из всех элементов были удалены все «чистые» пропуски, в том числе пробелы между различными элементами внутри элемента `<listing>`: `<keyword>`, `<const>`, `<var>` и т. д. Везде, где между элементами присутствуют непропускные символы (например, точка с запятой или круглая скобка), текстовый узел и все содержащиеся в нем пропуски сохраняются. Именно из-за этих текстовых узлов в результатах остаются некоторые пропуски. Если дойти до крайности и разместить каждый символ в отдельном элементе, весь текст в элементе `<listing>` сольется в одно целое.

Сравните с примером для элемента `<xsl:preserve-space>`.

<xsl:stylesheet>

Корневой элемент таблицы стилей XSLT. Идентичен элементу `<xsl:transform>`, включенному в спецификацию XSLT по историческим причинам.

Категория

Содержит всю таблицу стилей.

Обязательные атрибуты

`version`

Определяет версию XSLT, необходимую для обработки таблицы стилей. Для XSLT 1.0 значение всегда равно 1.0. По мере определения более поздних версий спецификации XSLT будут одновременно определяться необходимые значения атрибута `version`.

[2.0] В XSLT 2.0 атрибут должен быть равен `2.0`. Спецификация XSLT 2.0 определяет более жесткие правила обработки атрибута `version`, чем спецификация XSLT 1.0. Чтобы избежать осложнений, используйте только значения `1.0` и `2.0`.

XSLT 2.0 позволяет также задать атрибут `version` для любого элемента XSLT. Если вы хотите, чтобы какая-то часть таблицы стилей обрабатывалась по правилам версии 1.0, укажите атрибут `version="1.0"` для этой части таблицы стилей. Например, поведение элемента `<xsl:decimal-format>` в XSLT 2.0 изменилось, поэтому для обработки элемента по правилам XSLT 1.0 следует использовать запись `<xsl:decimal-format version="1.0">`.

`xmlns:xsl`

Определяет URI-адрес пространства имен XSL. В XSLT 1.0 и XSLT 2.0 этот атрибут должен содержать значение `http://www.w3.org/1999/XSL/Transform`. (Фактически это объявление пространства имен, а не атрибут, но оно является обязательным – без этого объявления таблица стилей работать не будет.)

Необязательные атрибуты

`id`

Определяет идентификатор таблицы стилей.

`extension-element-prefixes`

Определяет префиксы пространств имен, используемые для активизации элементов расширения. Префиксы в списке разделяются пробелами.

`exclude-result-prefixes`

Перечисляет префиксы пространств имен, которые не должны копироваться в выходной документ. Префиксы в списке разделяются пробелами.

[2.0] `xpath-default-namespace`

Определяет пространство имен, используемое функциями и операторами XPath по умолчанию. При преобразовании документа, использующего пространство имен по умолчанию (например, `http://www.oreilly.com`), имена всех элементов в выражениях XPath обычно должны уточняться пространствами имен. Определение `xpath-default-namespace="http://www.oreilly.com"` приказывает XPath использовать указанное пространство имен по умолчанию. Атрибут позволяет использовать разные пространства имен по умолчанию в разных частях таблицы стилей.

[2.0] `default-validation`

Определяет значение по умолчанию для атрибута `validation` элементов `<xsl:document>`, `<xsl:element>`, `<xsl:attribute>`, `<xsl:copy>`, `<xsl:copy-of>` и `<xsl:result-document>`. Допустимые значения атрибута: `preserve` и `strip`. Если атрибут отсутствует, по умолчанию используется значение `strip`.

[2.0] `default-collation`

Серия разделенных пробелами URI-адресов, определяющих последовательность упорядочения по умолчанию. Последовательность упорядочения по умолчанию используется элементами `<xsl:key>` и `<xsl:for-each-group>`, но она не влияет на упорядочение, используемое `<xsl:sort>`. Способ определения последовательностей упорядочения зависит от реализации; обращайтесь к документации своего процессора.

[2.0] `input-type-annotations`

Определяет, должна ли информация о типе данных сохраняться в выводе. Допустимые значения атрибута: `preserve`, `strip` и `unspecified`; по умолчанию используется значение `unspecified`.

Содержимое

Элемент содержит всю таблицу стилей. Дочерними элементами `<xsl:stylesheet>` могут быть следующие элементы:

`<xsl:attribute-set>`

[2.0] **[2.0]** `<xsl:character-map>`

`<xsl:decimal-format>`

[2.0] **[2.0]** `<xsl:function>`

`<xsl:import>`

[2.0] **[2.0 – Schema]** `<xsl:import-schema>`

`<xsl:include>`

```
<xsl:key>
<xsl:namespace-alias>
<xsl:output>
<xsl:param>
<xsl:preserve-space>
<xsl:strip-space>
<xsl:template>
<xsl:variable>
```

Местонахождение

Нет. `<xsl:stylesheet>` является корневым элементом таблицы стилей.

Где определяется

[1.0] XSLT, раздел 2.2 «Stylesheet Element».

[2.0] XSLT, раздел 3.6 «Stylesheet Element».

Пример

В качестве простого примера мы рассмотрим документ Hello World из спецификации XML 1.0:

```
<?xml version="1.0"?>
<greeting>
  Hello, World!
</greeting>
```

Документ преобразуется по следующей таблице стилей:

```
<?xml version="1.0"?>
<!-- stylesheet.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <xsl:apply-templates select="greeting"/>
  </xsl:template>

  <xsl:template match="greeting">
    <html>
      <body>
        <h1>
          <xsl:value-of select="."/>
        </h1>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Результат преобразования:

```
<html>
<body>
```

```
<h1>
  Hello, World!
</h1>
</body>
</html>
```

[2.0] **В XSLT 2.0 появился атрибут** `xpath-default-namespace`**, который упрощает работу с документами, обладающими пространством имен по умолчанию. Измененная версия документа с данными о продажах шоколада, использующего пространство имен по умолчанию:**

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate-default-namespace.xml -->
<report month="8" year="2006"
  xmlns="http://www.oreilly.com">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Простая таблица стилей выводит данные о продажах по каждому бренду:

```
<?xml version="1.0"?>
<!-- stylesheet2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Here are this month's sales figures:&#xA;&#xA;</xsl:text>
    <xsl:for-each select="/report/brand">
      <xsl:value-of select="name"/>
      <xsl:text>&#x9;</xsl:text>
      <xsl:value-of select="units"/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Однако результат разочаровывает:

```
Here are this month's sales figures:
```

Мы не получили никаких результатов, потому что не указали пространство имен элементов в выражениях XPath. В элемент `<xsl:stylesheet>` необходимо включить атрибут `xpath-default-namespace`:

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xpath-default-namespace="http://www.oreilly.com">
```

На этот раз мы получаем желаемый результат:

```
Here are this month's sales figures:

Lindt   27408
Callebaut       8203
Valrhona        22101
Perugina        14336
Ghirardelli     19268
```

<xsl:template>

Определяет выходной шаблон. Шаблоны, начинающиеся с `<xsl:template match="x"`, определяют преобразование для заданного элемента. Шаблоны, начинающиеся с `<xsl:template name="x"`, определяют набор выходных элементов, которые обрабатываются при обращении к данному шаблону по имени. У всех элементов `<xsl:template>` должен быть определен атрибут `match` или `name`. Элементы `<xsl:template>`, содержащие оба атрибута, тоже возможны, хотя встречаются довольно редко.

Категория

Элемент верхнего уровня.

Обязательные атрибуты

Нет; однако шаблон `<xsl:template>` должен содержать или атрибут `match`, или атрибут `name`, или оба атрибута одновременно.

Необязательные атрибуты

`match`

Описание элементов, для которых должен вызываться данный шаблон. Например, элемент `<xsl:template match="xyz">` определяет шаблон для обработки элементов `<xyz>`.

`name`

Определяет имя шаблона. Именованные шаблоны активизируются элементом `<xsl:call-template>`.

`mode`

Атрибут, определяющий режим обработки шаблона. Эта удобная синтаксическая конструкция позволяет создавать специализированные шаблоны для конкретных целей. Например, можно написать элемент `<xsl:template>` с атрибутом `mode="toc"` для построения сводки содержимого документа, а затем добавить к нему другие

элементы <xsl:template> с атрибутами mode="print", mode="online", mode="index" и т. д., чтобы обработать ту же информацию для других целей.

[2.0] В XSLT 2.0 атрибут mode может принимать значение #default, означающее, что шаблон используется в режиме по умолчанию, и значение #all для шаблонов, применимых во всех режимах. XSLT 2.0 позволяет также задать несколько имен режимов, разделенных пропусками. Попытка использования атрибута mode в шаблоне, не имеющем атрибута match, является ошибкой.

priority

Определяет числовой приоритет шаблона. Принимает любые числовые значения, кроме Infinity. Если процессор XSLT не может определить, какой шаблон следует использовать (иначе говоря, если несколько шаблонов обладают одинаковыми приоритетами по умолчанию), атрибут priority позволяет разрешить конфликт. Помните, что вы можете просто использовать атрибут priority для увеличения приоритета данного шаблона.

[2.0] В XSLT 2.0 использование атрибута priority в именованных шаблонах является ошибкой.

[2.0] as

Определяет тип данных результата шаблона. Если атрибут не указан, шаблон может возвращать любой тип данных.

Содержимое

Шаблон XSLT.

Местонахождение

<xsl:template> относится к элементам верхнего уровня, а его родителем может быть только элемент <xsl:stylesheet>.

Где определяется

[1.0] XSLT, раздел 5.3 «Defining Template Rules».

[2.0] XSLT, раздел 6.1 «Defining Template Rules».

Пример

В данном примере используется документ Hello World из спецификации XML 1.0:

```
<?xml version="1.0"?>
<greeting>
  Hello, World!
</greeting>
```

Документ преобразуется по следующей таблице стилей:

```
<?xml version="1.0"?>
<!-- stylesheet.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <xsl:apply-templates select="greeting"/>
  </xsl:template>
```

```
  <xsl:template match="greeting">
    <html>
      <body>
        <h1>
          <xsl:value-of select="."/>
        </h1>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Эта простая таблица стилей содержит два шаблона. Один из них соответствует корневому узлу (или узлу документа в XSLT 2.0), а другой – элементу `<greeting>` (`match="greeting"`).

```
<html>
<body>
<h1>
  Hello, World!
</h1>
</body>
</html>
```

<xsl:text>

Позволяет записывать литеральный текст в выходной документ. Главное преимущество элемента `<xsl:text>` заключается в том, что он предоставляет полный контроль над пропусками в выходном потоке.

Категория

Инструкция.

Обязательные атрибуты

Нет.

Необязательный атрибут

`disable-output-escaping`

Определяет, должны ли специальные символы экранироваться при записи в выходной документ. Например, если литеральный текст содержит символ >, обычно он записывается в выходной документ в виде `>`. При включенном атрибуте `disable-output-escaping="yes"` в этом нет необходимости, достаточно записать символ >. При использовании метода `<xsl:output method="text">` данный атрибут игнорируется, потому что при выводе текста экранирование не выполняется.

[2.0] В XSLT 2.0 этот атрибут считается устаревшим. Вместо него следует использовать новый элемент *[2.0]* `<xsl:character-map>`.

Содержимое

Литеральный текст и ссылки на сущности (например, `
`). В совокупности они обозначаются термином *PCDATA* (Parsed Character Data).

Местонахождение

Элемент <xsl:text> находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 7.2 «Creating Text».

[2.0] XSLT, раздел 11.4.2 «Creating Text Nodes Using xsl:text».

Пример

Следующая таблица стилей генерирует текст при помощи элементов <xsl:text>. Комбинация элементов <xsl:text> и <xsl:value-of> используется для создания связного предложения. В приведенном примере просто генерируется текстовый документ, но этот метод с таким же успехом может использоваться для создания текста элементов HTML и XML. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- text.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Test of &lt;xsl:text&gt;</title>
      </head>
      <body style="font-family: sans-serif;">
        <!-- По умолчанию disable-output-escaping="no" -->
        <h1>Test of &lt;xsl:text&gt;</h1>
        <p>
          <xsl:text>Your document contains </xsl:text>
          <xsl:value-of select="count(//*)"/>
          <xsl:text> elements and </xsl:text>
          <xsl:value-of select="count(//@*)"/>
          <xsl:text> attributes.  </xsl:text>
        </p>
        <p>
          <xsl:text
            disable-output-escaping="yes">&lt;Have a great day!&gt;</xsl:text>
        </p>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Для следующего документа XML:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
```

```
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

таблица стилей строит такой HTML-документ:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Test of &lt;xsl:text&gt;</title>
   </head>
   <body style="font-family: sans-serif;">
      <h1>Test of &lt;xsl:text&gt;</h1>
      <p>Your document contains 9 elements and 5 attributes.  </p>
      <p><Have a great day!></p>
   </body>
</html>
```

При просмотре документа в броузере неизвестный элемент `<Have>` во втором абзаце игнорируется. Атрибут `disable-output-escaping` позволяет сгенерировать текст, который не является действительным (или хотя бы корректным (well-formed)) кодом HTML, XML и XHTML. В таблицах стилей XSLT 1.0 эту возможность следует применять только в крайнем случае.

<xsl:transform>

Синоним `<xsl:stylesheet>`. Элемент был включен в спецификацию XSLT 1.0 по историческим причинам. По составу атрибутов, содержимому и прочим свойствам он почти не отличается от `<xsl:stylesheet>`. За дополнительной информацией обращайтесь к описанию элемента `<xsl:stylesheet>`.

Категория

Инструкция.

<xsl:value-of>

Вычисляет значение выражения XPath, преобразует полученное значение в текстовый узел и записывает его в дерево результата.

Категория

Инструкция.

Обязательный атрибут

`select`

Выражение XPath, результат которого вычисляется и записывается в выходной документ.

[2.0] В XSLT 2.0 этот атрибут не является обязательным. Элемент `<xsl:value-of>` должен иметь либо атрибут `select`, либо содержимое. Их одновременное присутствие (или, наоборот, отсутствие) является ошибкой.

Необязательные атрибуты

`disable-output-escaping`

Определяет, должны ли специальные символы экранироваться при записи в выходной документ. Например, если литеральный текст содержит символ >, обычно он записывается в выходной документ в виде `>`. При включенном атрибуте `disable-output-escaping="yes"` в этом нет необходимости, достаточно записать символ >. Процессор XSLT использует данный атрибут только для методов вывода `html` и `xml`. При использовании метода `<xsl:output method="text">` этот атрибут игнорируется, потому что при выводе текста экранирование не выполняется. Атрибут `disable-output-escaping` более подробно рассматривается в описании элемента `<xsl:text>`.

[2.0] В XSLT 2.0 этот атрибут считается устаревшим. Вместо него следует использовать карту символов; за дополнительной информацией обращайтесь к описанию элемента *[2.0]* `<xsl:character-map>`.

[2.0] `separator`

Определяет символы для разделения множественных значений, генерируемых атрибутом `select`. По умолчанию в качестве разделителя используется пробел (`#x20`). Значение атрибута `separator` выводится после каждого значения в списке, кроме последнего.

В XSLT 1.0 элемент `<xsl:value-of>` выбирает первый объект набора узлов и удаляет все остальные. В XSLT 2.0 выводятся все узлы, разделенные значениями атрибута `separator`.

Содержимое

Нет. `<xsl:value-of>` является пустым элементом.

[2.0] В XSLT 2.0 элемент `<xsl:value-of>` может иметь содержимое; оно обрабатывается и записывается в выходной поток.

Местонахождение

Элемент `<xsl:value-of>` находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 7.6.1 «Generating Text with xsl:value-of».

[2.0] XSLT, раздел 11.4.3 «Generating Text with xsl:value-of».

Пример

В следующем примере элемент `<xsl:value-of>` генерирует некоторый текст. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- value-of.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Your document contains </xsl:text>
```

```
    <xsl:value-of select="count(//*)"/>
    <xsl:text> elements and </xsl:text>
    <xsl:value-of select="count(//@*)"/>
    <xsl:text> attributes.&#xA;Have a great day!</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Входные данные берутся из следующего документа XML:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Результат:

```
Your document contains 9 elements and 5 attributes.
Have a great day!
```

[2.0] Прежде чем расстаться с элементом `<xsl:value-of>`, мы рассмотрим его новые возможности в XSLT 2.0. В следующей короткой таблице стилей используются непустой элемент `<xsl:value-of>` и атрибут `separator`:

```
<?xml version="1.0"?>
<!-- value-of2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:value-of>
      <xsl:text>Here is a list of the </xsl:text>
      <xsl:value-of select="count(//*)"/>
      <xsl:text> elements in your document:&#xA;&#xA;</xsl:text>
    </xsl:value-of>
    <xsl:value-of select="//*/name()" separator="&#xA;"/>
  </xsl:template>

</xsl:stylesheet>
```

При обработке файла *albums.xml* с новой таблицей стилей получается такой результат:

```
Here is a list of the 9 elements in your document:

list
title
listitem
listitem
listitem
listitem
```

```
listitem
listitem
listitem
```

Первый абзац генерируется элементом <xsl:value-of> с тремя элементами, одним из которых является другой элемент <xsl:value-of>. Остальная часть документа генерируется одним элементом <xsl:value-of> с удобным атрибутом separator. Чтобы сделать то же самое в XSLT 1.0, нам пришлось бы заменить строку:

```
<xsl:value-of select="//*/name()" separator="&#xA;"/>
```

более сложным фрагментом разметки:

```
<xsl:for-each select="//*">
  <xsl:value-of select="name()"/>
  <xsl:if test="position() != last()">
    <xsl:text>&#xA;</xsl:text>
  </xsl:if>
</xsl:for-each>
```

<xsl:variable>

Определяет переменную. Если элемент <xsl:variable> находится в элементе верхнего уровня, он определяет глобальную переменную, доступную в любой точке таблицы стилей. В противном случае определяется локальная переменная, существующая только в границах элемента, содержащего <xsl:variable>. Значение переменной может определяться одним из двух способов: оно либо задается в атрибуте select, либо определяется в шаблоне XSLT внутри самого элемента <xsl:variable>. Если ни один из этих способов не используется, значением переменной является пустая строка.

Категория

Элемент верхнего уровня или инструкция.

Обязательный атрибут

name

Определяет имя переменной.

Необязательные атрибуты

select

Выражение XPath, определяющее значение переменной.

[2.0 – Schema] as

Тип данных переменной. Например, элемент <xsl:variable name="age" as="xs:integer" select="11"/> определяет новую переменную типа xs:integer. Если значение не удается преобразовать к заданному типу, происходит ошибка; например, включение атрибута select="'really, really old'" приведет к ошибке. При использовании схемосовместимого процессора XSLT можно указать пользовательский тип данных.

Содержимое

Элемент <xsl:variable> либо пуст, либо содержит шаблон XSLT. Одновременное наличие содержимого и атрибута select является фатальной ошибкой.

Местонахождение

Элемент <xsl:stylesheet> является элементом верхнего уровня или находится в шаблоне.

Где определяется

[1.0] XSLT, раздел 11 «Variables and Parameters».

[2.0] XSLT, раздел 9 «Variables and Parameters».

Пример

Следующая таблица стилей определяет несколько переменных:

```
<?xml version="1.0"?>
<!-- variable.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:variable name="favoriteNumber" select="23"/>
  <xsl:variable name="favoriteColor" select="'blue'"/>
  <xsl:variable name="complicatedVariable">
    <xsl:choose>
      <xsl:when test="count(//listitem) > 10">
        <xsl:text>really long list</xsl:text>
      </xsl:when>
      <xsl:when test="count(//listitem) > 5">
        <xsl:text>moderately long list</xsl:text>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>fairly short list</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <xsl:template match="/">
    <xsl:text>Hello!  Your favorite number is </xsl:text>
    <xsl:value-of select="$favoriteNumber"/>
    <xsl:text>.&#xA;Your favorite color is </xsl:text>
    <xsl:value-of select="$favoriteColor"/>
    <xsl:text>.&#xA;&#xA;Here is a </xsl:text>
    <xsl:value-of select="$complicatedVariable"/>
    <xsl:text>:&#xA;</xsl:text>
    <xsl:variable name="listitems" select="list/listitem"/>
    <xsl:call-template name="processListItems">
      <xsl:with-param name="items" select="$listitems"/>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="processListItems">
    <xsl:param name="items"/>
    <xsl:variable name="favoriteColor">
      <xsl:text>chartreuse</xsl:text>
    </xsl:variable>

    <xsl:text>    (Your favorite color is now </xsl:text>
    <xsl:value-of select="$favoriteColor"/>
    <xsl:text>.)&#xA;</xsl:text>
```

```
    <xsl:for-each select="$items">
      <xsl:value-of select="position()"/>
      <xsl:text>.  </xsl:text>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей используется для преобразования следующего документа:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Результат преобразования:

```
Hello!  Your favorite number is 23.
Your favorite color is blue.

Here is a moderately long list:
    (Your favorite color is now chartreuse.)
1.  The Sacred Art of Dub
2.  Only the Poor Man Feel It
3.  Excitable Boy
4.  Aki Special
5.  Combat Rock
6.  Talking Timbuktu
7.  The Birth of the Cool
```

В XSLT 1.0 существование двух одноименных переменных на одном уровне является ошибкой. Кроме того, ошибкой также считается существование одноименных элементов `<xsl:variable>` и `<xsl:param>` на одном уровне. В XSLT 2.0 ни одна из этих ситуаций не является ошибкой.

Следует обратить внимание на некоторые обстоятельства в нашей таблице стилей. Прежде всего, при определении значений первых двух переменных (`favoriteNumber` и `favoriteColor`) нам пришлось заключить в апострофы строку `blue`, тогда как делать подобное со строкой `23` было не нужно. Если не заключить в апострофы `blue`, процессор XSLT решит, что речь идет о наборе узлов (или последовательности) всех элементов `<blue>` в контекстном узле. Разумеется, это будет совсем не то, что нам требуется.

Учтите, что апострофы изменяют интерпретацию числовых значений. Запись `35` представляет числовое значение (число в XSLT 1.0, `xs:integer` в XSLT 2.0), тогда как значение `'35'` представляет *строку* `35`. На первый взгляд это различие кажется несущественным, но оно серьезно влияет на работу таблиц стилей, особенно в XSLT 2.0.

Стоит также заметить, что в таблице стилей определяются две переменные с именем favoriteColor. Одна из них – глобальная, поскольку ее родителем является элемент <xsl:stylesheet>; другая – локальная, потому что она определяется в элементе <xsl:template>. При обращении к favoriteColor в шаблоне match="/" переменная имеет одно значение, а при обращении в шаблоне name="processListItems" – другое.

Использование элемента <xsl:choose> для инициализации <xsl:variable> – стандартный прием. Этот прием эквивалентен следующей программной конструкции:

```
String complicatedVariable;

if (count(listitems) > 10)
  complicatedVariable = "really long list";
else if (count(listitems)) > 5)
  complicatedVariable = "moderately long list";
else
  complicatedVariable = "fairly short list";
```

Переменная может относиться к любому из типов переменных XPath или XSLT, включая *[1.0]* набор узлов или *[2.0]* последовательность. При вызове шаблона processListItems в качестве параметра передается переменная с набором узлов всех элементов <listitem> в нашем документе. Внутри шаблона processListItems наша переменная (формально ставшая параметром) используется в элементе <xsl:for-each>.

[2.0] В последнем примере атрибут as демонстрирует использование типов данных. Простая таблица стилей:

```
<?xml version="1.0"?>
<!-- variable2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:variable name="numberOne" as="xs:integer" select="23"/>
  <xsl:variable name="numberTwo" as="xs:double" select="2.718281828459"/>
  <xsl:variable name="numberThree" as="xs:float" select="$numberTwo"/>
  <xsl:variable name="numberFour" as="xs:integer"
    select="xs:integer($numberTwo)"/>
  <xsl:variable name="dateValue" as="xs:date" select="xs:date('1995-04-21')"/>
  <xsl:variable name="whatever" as="xs:integer" select="xs:integer(42)"/>

  <xsl:template match="/">
    <xsl:call-template name="bob">
      <xsl:with-param name="whatever" select="xs:integer(8)"/>
    </xsl:call-template>
    <xsl:text>&#xA;whatever&#x9;</xsl:text>
    <xsl:value-of select="$whatever"/>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

  <xsl:template name="bob">
    <xsl:param name="whatever" as="xs:integer"/>
    <xsl:variable name="whatever" as="xs:integer" select="xs:integer(4)"/>
    <xsl:variable name="whatever" as="xs:integer" select="xs:integer(98)"/>
    <xsl:text>Hello!  The values of your variables are:&#xA;</xsl:text>
    <xsl:text>&#xA;numberOne&#x9;</xsl:text>
    <xsl:value-of select="$numberOne"/>
    <xsl:text>&#xA;numberTwo&#x9;</xsl:text>
```

```
    <xsl:value-of select="$numberTwo"/>
    <xsl:text>&#xA;numberThree&#x9;</xsl:text>
    <xsl:value-of select="$numberThree"/>
    <xsl:text>&#xA;numberFour&#x9;</xsl:text>
    <xsl:value-of select="$numberFour"/>
    <xsl:text>&#xA;dateValue&#x9;</xsl:text>
    <xsl:value-of select="$dateValue"/>
    <xsl:text>&#xA;whatever&#x9;</xsl:text>
    <xsl:value-of select="$whatever"/>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

При обработке таблицы стилей будет получен следующий результат:

```
Hello!  The values of your variables are:

numberOne       23
numberTwo       2.718281828459
numberThree     2.7182817
numberFour      2
dateValue       1995-04-21
whatever        98

whatever        42
```

Обратите внимание на использование конструктора `xs:date` для создания действительной даты и на потерю точности при копировании значения `numberTwo` типа `double` в переменную `numberThree` типа `float`. Для преобразования значения `numberTwo` типа `double` к типу `integer`, необходимому для `numberFour`, приходится использовать конструктор `xs:integer`.

Все элементы `<xsl:variable>` инициализируются литеральными значениями; если заменить их чем-то недействительным (например, присвоить переменной `$numberOne` типа `xs:integer` значение `'blue'`), таблица стилей работать не будет. Если значения переменных задаются на стадии выполнения, попытка задать переменной недействительное значение приведет к фатальной ошибке.

В представленной таблице стилей несколько раз используется переменная `$whatever`. Так как в XSLT 2.0 переменные и параметры могут временно замещать друг друга, значением `$whatever` будет последнее значение, определенное в текущем контексте. В именованном шаблоне переменная `$whatever` равна `98` — последнему значению, ассоциированную с этим именем. То, что переменной `$whatever` ранее присваивались значения `4` (как локальной переменной), `8` (как параметру) и `42` (как глобальной переменной), роли не играет. Когда таблица стилей возвращает управление вызывающему шаблону, значение `$whatever` становится равным `42`. Параметр, переданный именованному шаблону, и локальные переменные внутри этого шаблона вышли из области видимости.

При работе с переменными в XSLT 2.0 существует один нюанс: если создать переменную, содержащую узлы, и задать ее тип данных атрибутом `as`, то переменная будет работать так, как положено. Но если атрибут `as` не задан, переменная создается как новый узел документа с дочерними узлами, определенными в переменной. Эта тема более подробно рассматривается в разделе 9.4 спецификации XSLT 2.0. В общем случае рекомендуется использовать атрибут `as`.

<xsl:when>

Определяет одну ветвь элемента <xsl:choose>. В XSLT элемент <xsl:choose> эквивалентен конструкции if-then-else в традиционных языках программирования.

Категория

Вложенная инструкция (<xsl:when> всегда является частью элемента <xsl:choose>).

Обязательный атрибут

test

Содержит проверяемое логическое выражение. Если результат вычисления выражения равен true, обрабатывается содержимое элемента <xsl:when>; в противном случае содержимое <xsl:when> игнорируется. При обработке элемента <xsl:choose> процессор XSLT перебирает все элементы <xsl:when>, пока не найдет элемент, у которого вычисление атрибута test дает результат true. Найденный элемент <xsl:when> обрабатывается, а все последующие элементы <xsl:when> игнорируются. В элементе <xsl:choose> всегда обрабатывается не более одного элемента <xsl:when>.

Необязательные атрибуты

Нет.

Содержимое

Шаблон XSLT.

Местонахождение

Только внутри элемента <xsl:choose>.

Где определяется

[1.0] XSLT, раздел 9.2 «Conditional Processing with xsl:choose».

[2.0] XSLT, раздел 8.2 «Conditional Processing with xsl:choose».

Пример

В следующем примере <xsl:choose> используется для выбора цвета фона в строках таблицы HTML. Таблица стилей перебирает четыре возможных значения, а элемент <xsl:when> определяет значение атрибута style в генерируемом документе HTML. Таблица стилей обрабатывает следующий документ XML:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en_GB">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Таблица стилей:

```
<?xml version="1.0"?>
<!-- when.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>
          <xsl:value-of select="list/title"/>
        </title>
      </head>
      <body style="font-family: sans-serif; color: white;">
        <h1 style="color: black;">
          <xsl:value-of select="list/title"/>
        </h1>
        <table border="1" cellpadding="5"
          style="font-weight: bold;">
          <xsl:for-each select="list/listitem">
            <tr>
              <td>
                <xsl:attribute name="style">
                  <xsl:choose>
                    <xsl:when test="position() mod 4 = 0">
                      <xsl:text>background: yellow; color: black;</xsl:text>
                    </xsl:when>
                    <xsl:when test="position() mod 4 = 1">
                      <xsl:text>background: blue;</xsl:text>
                    </xsl:when>
                    <xsl:when test="position() mod 4 = 2">
                      <xsl:text>background: white; color: black;</xsl:text>
                    </xsl:when>
                    <xsl:otherwise>
                      <xsl:text>background: black;</xsl:text>
                    </xsl:otherwise>
                  </xsl:choose>
                </xsl:attribute>
                <xsl:value-of select="."/>
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Сгенерированный документ HTML перебирает разные цвета фона:

```
<html>
  <head>
    <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Albums I've bought recently:</title>
  </head>
```

```
    <body style="font-family: sans-serif; color: white;">
      <h1 style="color: black;">Albums I've bought recently:</h1>
      <table border="1" cellpadding="5" style="font-weight: bold;">
        <tr>
          <td style="background: blue;">The Sacred Art of Dub</td>
        </tr>
        <tr>
          <td style="background: white; color: black;">Only the Poor Man Feel It</td>
        </tr>
        <tr>
          <td style="background: black;">Excitable Boy</td>
        </tr>
        <tr>
          <td style="background: yellow; color: black;">Aki Special</td>
        </tr>
        ...
      </table>
    </body>
  </html>
```

Ячейки таблицы со стилями `"background: yellow; color: black;"`, `"background: blue;"` и `"background: white; color: black;"` генерируются элементами `<xsl:when>`, а стиль `"background: black;"` генерируется элементом `<xsl:otherwise>`.

На рис. А.17 показано, как сгенерированный документ HTML выглядит в броузере.

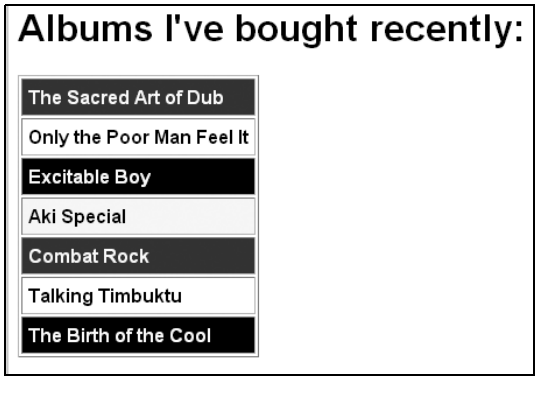


***Рис. А.17.** Использование элемента <xsl:when> для перебора цветов фона*

<xsl:with-param>

Определяет параметр, передаваемый шаблону. При вызове шаблона передаются значения определяемых в нем параметров.

Категория

Вложенная инструкция; `<xsl:with-param>` всегда находится внутри элемента `<xsl:apply-templates>` или `<xsl:call-template>`.

[2.0] В XSLT 2.0 элемент может также находиться в новых элементах `<xsl:apply-imports>` и `<xsl:next-match>`.

Описание

Элемент <xsl:with-param> определяет параметр, передаваемый шаблону. При вызове шаблона передаются значения определяемых в нем параметров. Значение параметра определяется одним из трех способов:

- Если элемент <xsl:with-param> пуст и не содержит атрибута select, значение шаблону не передается.
- Если элемент <xsl:with-param> пуст, но содержит атрибут select, значение параметра определяется значением атрибута select.
- Если элемент <xsl:with-param> не пуст, значение параметра определяется результатом обработки его содержимого.

Если шаблон вызывается без передачи значения (<xsl:with-param name="x"/>), то используется значение параметра по умолчанию (если оно имеется). Значение параметра по умолчанию определяется в элементе <xsl:param> внутри самого элемента <xsl:template>; за подробностями обращайтесь к описанию элемента <xsl:param>.

Обязательный атрибут

name

Определяет имя параметра.

Необязательные атрибуты

select

Выражение XPath, определяющее значение параметра. Одновременное наличие у <xsl:with-param> содержимого и атрибута select является фатальной ошибкой.

[2.0 – Schema] as

Определяет тип данных параметра. Им может быть любой из встроенных типов данных или же тип данных, определенный в схеме (при использовании схемосовместимого процессора XSLT 2.0).

[2.0] tunnel

Определяет, является ли параметр туннельным. Допустимые значения: yes и no, по умолчанию используется значение no.

Содержимое

Элемент <xsl:with-param> либо пуст, либо содержит шаблон XSLT. Одновременное наличие содержимого и атрибута select является фатальной ошибкой.

Местонахождение

Элементы <xsl:apply-templates>, <xsl:call-template>, *[2.0]* <xsl:apply-imports> и *[2.0]* <xsl:next-match>.

Где определяется

[1.0] XSLT, раздел 11.6 «Passing Parameters to Templates».

[2.0] XSLT, раздел 10.1.1 «Passing Parameters to Templates».

Пример

В следующем примере определяется таблица стилей с параметрами. Некоторые параметры являются глобальными и определяются за пределами таблицы стилей:

```
<?xml version="1.0"?>
<!-- with-param.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:param name="favoriteNumber" select="23"/>
  <xsl:param name="favoriteColor"/>

  <xsl:template match="/">
    <xsl:text>&#xA;</xsl:text>
    <xsl:value-of select="list/title"/>
    <xsl:text>&#xA;</xsl:text>
    <xsl:variable name="listitems" select="list/listitem"/>
    <xsl:call-template name="processListItems">
      <xsl:with-param name="items" select="$listitems"/>
      <xsl:with-param name="color" select="'yellow'"/>
      <xsl:with-param name="number" select="$favoriteNumber"/>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="processListItems">
    <xsl:param name="items"/>
    <xsl:param name="color" select="'blue'"/>

    <xsl:for-each select="$items">
      <xsl:value-of select="position()"/>
      <xsl:text>.  </xsl:text>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>

    <xsl:text>&#xA;</xsl:text>

    <xsl:text>Your favorite color is </xsl:text>
    <xsl:value-of select="$favoriteColor"/>
    <xsl:text>.</xsl:text>
    <xsl:text>&#xA;</xsl:text>
    <xsl:text>The color passed to this template is </xsl:text>
    <xsl:value-of select="$color"/>
    <xsl:text>.</xsl:text>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей используется для преобразования следующего документа:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Таблица стилей содержит два глобальных параметра (favoriteNumber и favoriteColor), причем для favoriteNumber определяется значение по умолчанию. Таблица стилей также передает параметр из шаблона match="/" шаблону name="processListItems"; этот параметр содержит набор узлов. Результат преобразования:

```
Albums I've bought recently:
1.  The Sacred Art of Dub
2.  Only the Poor Man Feel It
3.  Excitable Boy
4.  Aki Special
5.  Combat Rock
6.  Talking Timbuktu
7.  The Birth of the Cool

Your favorite color is orange.
The color passed to this template is yellow.
```

Обратите внимание: при вызове шаблона processListItems передаются три параметра, только два из которых определяются внутри шаблона. В XSLT 1.0 неопределенные параметры просто игнорируются; в XSLT 2.0 они приводят к фатальной ошибке. Если преобразовать таблицу стилей к виду <xsl:stylesheet version="2.0" ..., она не будет работать. Также обратите внимание на то, что параметр $items представляет собой набор узлов, поэтому он может использоваться в элементе <xsl:for-each>.

Для получения этого результата в Saxon используется следующая команда:

```
java net.sf.saxon.Transform albums.xml with-param.xsl favoriteColor=orange
```

В Xalan команда выглядит несколько иначе:

```
java org.apache.xalan.xslt.Process -in albums.xml -xsl with-param.xsl
  -param favoriteColor orange
```

(Команда должна вводиться в одной строке.) За подробной информацией о глобальных параметрах и передаче их в разных процессорах XSLT обращайтесь к разделу «Глобальные параметры» главы 5.

[2.0 – Schema] Завершая описание <xsl:with-param>, мы рассмотрим пример с поддержкой схем. В этом примере используется параметр с типом данных zipcode. Документ XML Schema с определением типа данных выглядит так:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- zip.xsd -->
<xsd:schema
  xmlns="http://www.oreilly.com/xslt/zip"
  targetNamespace="http://www.oreilly.com/xslt/zip"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:simpleType name="zipcode">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Таблица стилей, проверяющая действительность zipcode, выглядит так:

```
<?xml version="1.0"?>
<!-- with-param2.xsl -->
<xsl:stylesheet version="2.0"
```

```
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:zip="http://www.oreilly.com/xslt/zip"
    xmlns="http://www.oreilly.com/xslt/zip"

    xmlns:po="http://www.oreilly.com/xslt">

    <xsl:import-schema namespace="http://www.oreilly.com/xslt/zip"
      schema-location="zip.xsd" />

    <xsl:import-schema namespace="http://www.oreilly.com/xslt"
      schema-location="po.xsd" />

    <xsl:output method="text"/>

    <xsl:template match="/">
      <xsl:choose>
        <xsl:when
          test="/po:purchase-order/po:customer/po:address/po:zip
                castable as zip:zipcode">
          <xsl:call-template name="postalCode">
            <xsl:with-param name="zip" as="zip:zipcode"
              select="/po:purchase-order/po:customer/po:address/po:zip"/>
          </xsl:call-template>
        </xsl:when>
        <xsl:otherwise>
          <xsl:text>The &lt;zip&gt; element isn't valid!&#xA;</xsl:text>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:template>

    <xsl:template name="postalCode">
      <xsl:param name="zip" as="zip:zipcode"/>
      <xsl:text>The value </xsl:text>
      <xsl:value-of select="$zip"/>
      <xsl:text> is a valid Zip code!&#xA;</xsl:text>
    </xsl:template>

  </xsl:stylesheet>
```

Применяя этот шаблон к действительному элементу `<po:purchase-order>` с элементом `<zip>`, который может быть преобразован в значение `zip:zipcode`, мы используем элемент `<xsl:with-param>` для передачи параметра типа `zip:zipcode`. Воспользуемся уже знакомым документом с заказом:

```
<?xml version="1.0"?>
<!-- good-po.xml -->
<purchase-order id="38292"   xmlns="http://www.oreilly.com/xslt">
  <date year="2001" month="6" day="19"/>
  <customer id="4738" level="Platinum">
    <address type="business">
      ...
      <zip>48392</zip>
    </address>
  </customer>
  <items>
  ...
  </items>
</purchase-order>
```

Результат выглядит весьма тривиально:

```
The value 48392 is a valid Zip code!
```

Обратите внимание: в таблице стилей импортируются две схемы – для типа данных `zip:zipcode` и для элемента `<po:purchase-order>`,– хотя они могли бы размещаться и в одном файле. Если бы элементы заказа принадлежали пространству имен, нам пришлось бы указывать префикс `po:` для всех элементов в заказе или использовать атрибут `xpath-default-namespace`. Оператор XPath `castable as` проверяет, что значение элемента `<po:zip>` может быть преобразовано в значение `zip:zipcode`. Таким образом, элементы `<xsl:with-param>` и `<xsl:param>` используются только при действительных данных. Если данные действительны, мы создаем параметр, зная, что требования атрибутов `as` выполняются.

И последнее: атрибут `as` может содержать только типы-последовательности. А конкретнее, атрибут `as` не может ссылаться на сложные типы, объявленные в XML Schema. Схему заказа можно переписать так, чтобы в ней появился сложный тип `purchase-order`. Чтобы использовать его в атрибуте `as`, закодируйте последний в виде `as="element(*, po:purchase-order)"`.

B

Справочник по XPath

В этом приложении содержится справочная информация из спецификации XPath о типах узлов, осях, операторах и типах данных.

Типы узлов XPath

Узлы в XPath делятся на семь типов. (В XPath 2.0 они называются *разновидностями* (*kinds*) узлов.) Здесь мы ограничимся справочным материалом; за дополнительной информацией о разных типах узлов обращайтесь к разделу «Модель данных XPath» в главе 3.

Корневой узел

Корневой узел является корнем дерева. В отличие от других узлов, он не имеет родителя. Его дочерними узлами являются корневой элемент документа и все комментарии и инструкции по обработке, находящиеся за пределами элемента документа. Корневой узел не имеет расширенного (expanded) имени. В XPath 2.0 он называется *узлом документа*.

Узлы элементов

Каждый элемент в исходном документе XML представлен узлом элемента. Расширенное имя элемента представляет собой его локальное имя в сочетании с пространством имен, действующим для данного элемента. Для обращения к различным частям имени элемента используются функции `name()`, `local-name()` и `namespace-uri()`. Элемент из документа XML:

```
<xyz:report xmlns:xyz="http://www.xyz.com/">
```

Для приведенного узла элемента эти три функции возвращают следующие значения:

```
name()
    xyz:report
local-name()
```

```
      report
   namespace-uri()
      http://www.xyz.com/
```

Узлы атрибутов

Атрибуты элементов из документа XML в XPath представлены узлами атрибутов. По аналогии с узлами элементов узел атрибута обладает расширенным именем. Узлы атрибутов заданного узла элемента строятся на базе атрибутов, явно закодированных в элементе XML, и из атрибутов, определяемых со значениями по умолчанию в DTD.

В XPath используется подход, отличный от модели DOM: узел элемента является родителем всех своих атрибутов, хотя атрибуты не являются его дочерними узлами. Иначе говоря, при выборе всех дочерних узлов узла элемента узлы атрибутов, принадлежащих этому узлу элемента, не выбираются.

Текстовые узлы

Текстовые узлы просто содержат текст элемента. Если исходный текст в документе XML содержит символы или ссылки на сущности, они разрешаются до создания текстового узла XPath. Аналогично каждая существующая секция CDATA предстает как текстовый узел. Невозможно определить, существовала ли некоторая часть текстового узла ранее в виде символа, ссылки на сущность или секции CDATA.

Узлы комментариев

Узлы комментариев тоже очень просты; они содержат некоторый текст. Каждый комментарий в исходном документе (за исключением комментариев в DTD) преобразуется в узел комментария. Текст узла комментария (возвращаемый при выборке `<xsl:value-of select=".">`) содержит все символы, находящиеся внутри комментария, за исключением открывающего `<!--` и закрывающего `-->` маркеров.

Узлы инструкций по обработке

Узел инструкции по обработке состоит из двух частей: имени (возвращаемого функцией `name()`) и строкового значения. Строковое значение состоит из всех символов, следующих за именем, включая пропуски, но без маркера `?>`, закрывающего инструкцию по обработке.

Узлы пространств имен

Узлы пространств имен почти никогда не встречаются в таблицах стилей XSLT; они используются исключительно процессором XSLT. Следует помнить, что объявление пространства имен (например, `xmlns:auth="http://www.authors.net"`) выглядит как атрибут в исходном коде XML, но преобразуется в узел пространства имен, а не в узел атрибута. Узлы пространств имен создаются как для определяемых префиксов пространств имен, так и для всех пространств имен по умолчанию.

Тесты узлов XPath

В XPath определяется ряд тестов узлов, которые могут использоваться для выборки узлов из исходного дерева. Тесты узлов позволяют выбирать узлы, которые не могут

быть выбраны другими способами. (Хотя тесты узлов выглядят и работают как функции, с технической точки зрения они относятся к другой категории.) Существуют следующие тесты узлов:

`text()`

Выбирает все дочерние текстовые узлы контекстного узла.

`comment()`

Выбирает все дочерние узлы комментариев контекстного узла.

`processing-instruction()`

Выбирает все дочерние узлы инструкций по обработке контекстного узла. В отличие от других тестов узлов, `processing-instruction()` может получать необязательный аргумент; например, `processing-instruction('xml-stylesheet')` выбирает все инструкции по обработке с именем `xml-stylesheet`.

`node()`

Возвращает все узлы независимо от типа. При использовании этого теста выбираются все узлы элементов, узлы атрибутов, узлы инструкций по обработке и т. д. (Учтите, что применение `node()` к оси `child` *не* возвращает узлы атрибутов, потому что узлы атрибутов не считаются дочерними узлами.)

[2.0] `attribute()`

Возвращает любой атрибут. При использовании с именем атрибута (`attribute(public-domain)`) возвращает все атрибуты с заданным именем. При использовании с именем атрибута и типом данных (`attribute(public-domain, xs:boolean)`) возвращает все атрибуты с заданными именем и типом данных. Наконец, при использовании с метасимволом вместо имени атрибута (`attribute(*, xs:boolean)`) возвращает все атрибуты с заданным типом данных.

[2.0] `element()`

Возвращает любой элемент. При использовании с именем элемента (`element(author)`) возвращает все элементы с заданным именем. При использовании с именем элемента и типом данных (`element(year-of-birth, xs:gYear)`) возвращает все элементы с заданными именем и типом данных. Наконец, при использовании с метасимволом вместо имени элемента (`element(*, xs:gYear)`) возвращает все элементы с заданным типом данных.

[2.0] `schema-element(name)`

По имени элемента, глобально объявленному в XML Schema, возвращает элементы с такими же именем и типом данных, как у элемента, определенного в схеме. Также возвращает все элементы из группы подстановки заданного элемента. Группы подстановки более подробно рассматриваются в приложении D.

[2.0] `document-node()`

Представляет узлы документов. Тест узла может включать имя; например, тест `document-node(element(sonnet))` возвращает узел документа с корневым элементом `<sonnet>`.

[2.0] Хотя конструкция `item()` внешне похожа на тест узла, она используется только как тип данных. Например, переменная `<xsl:variable name="something" as="item()">` определяет переменную, которая может содержать единственное значение (любой узел или атомарный тип).

Оси XPath

Спецификация XPath определяет 13 разных осей, содержащих разные узлы. Конкретный набор узлов, входящих в заданную ось, зависит от контекстного узла. Далее перечислены все 13 осей; за более подробными описаниями обращайтесь к разделу «Модель данных XPath» главы 3.

`child`

Содержит дочерние узлы контекстного узла. Как уже упоминалось ранее, выражения XPath `child::lines/child::line` и `lines/line` эквивалентны. Если в выражении XPath (например, `sonnet`) ось не указана, по умолчанию используется ось `child`.

`parent`

Родитель контекстного узла, если он существует. (Если контекстный узел является корневым, ось `parent` возвращает пустой набор узлов.) В выражениях XPath ось `parent` может быть представлена сокращением `..`, обозначающим переход к родителю текущего узла. Если элементы `<first-name>` и `<last-name>` являются дочерними по отношению к элементу `<author>`, а элемент `<first-name>` является контекстным узлом, выражения `../last-name`, `parent::author/last-name` и `parent::*/last-name` эквивалентны. Если контекстный узел не имеет родителя, ось возвращает пустой набор узлов.

`self`

Содержит сам контекстный узел. В выражениях XPath ось `self` может быть представлена сокращением `.` (точка). Выражения `.`, `self::node()` и `self::*` в XSLT 1.0 эквивалентны.

[2.0] В XSLT 2.0 ось `self` выбирает контекстный *объект*, который может и не быть узлом. Если контекстный объект является атомарным значением, выражения `self::node()` и `self::*` приводят к выдаче ошибки процессором XSLT. В этом случае к оси `self` можно обратиться только с использованием точки. Если контекстный объект является узлом, ось `self` работает так же, как в XSLT 1.0.

`attribute`

Содержит атрибуты контекстного узла. Если контекстный узел не является узлом элемента, ось пуста. Ось `attribute` может быть представлена сокращенным обозначением `@`. Выражения `attribute::type` и `@type` эквивалентны.

`ancestor`

Содержит родителя контекстного узла, родителя его родителя и т. д. Ось `ancestor` всегда содержит корневой узел, если только контекстный узел не является корневым.

`ancestor-or-self`

Содержит контекстный узел, его родителя, родителя его родителя и т. д. Ось всегда содержит корневой узел.

`descendant`

Содержит всех потомков контекстного узла, всех потомков потомков контекстного узла и т. д. Помните, что ось `descendant` не включает узлы атрибутов и пространств имен. (Как упоминалось ранее, хотя узлы атрибутов имеют родительский узел элемента, узел атрибута не считается его дочерним узлом.)

`descendant-or-self`

Содержит контекстный узел, его потомков, потомков его потомков и т. д.

`preceding-sibling`

Содержит все предшествующие узлы одного уровня с контекстным узлом – иначе говоря, все узлы, имеющие общего родителя с контекстным узлом, но находящиеся перед ним в документе XML. Если контекстный узел является узлом атрибута или пространства имен, ось `preceding-sibling` пуста.

`following-sibling`

Содержит все последующие узлы одного уровня с контекстным узлом – иначе говоря, все узлы, имеющие общего родителя с контекстным узлом, но следующие после него в документе XML. Если контекстный узел является узлом атрибута или пространства имен, ось `following-sibling` пуста.

`preceding`

Содержит все узлы, предшествующие контекстному узлу в документе, кроме предков, узлов атрибутов и пространств имен.

`following`

Содержит все узлы, следующие за контекстным узлом в документе, кроме потомков, узлов атрибутов и пространств имен.

`namespace`

Содержит узлы пространств имен контекстного узла. Если контекстный узел не является узлом элемента, ось пуста.

[2.0] В XPath ось `namespace` считается устаревшей, и ее использование не рекомендуется.

Пять осей `ancestor`, `descendant`, `following`, `preceding` и `self` обеспечивают логическую разбивку всего содержимого документа XML (кроме узлов атрибутов и пространств имен). Любой узел в дереве XPath принадлежит к одной из пяти осей, причем эти оси не перекрываются.

Контекст XPath

Контекст в выражении XPath складывается из следующих компонентов:

Контекстный узел

Узел, обрабатываемый в настоящий момент. *[2.0]* Если контекстный объект является узлом, контекстный узел совпадает с контекстным объектом. Если контекстный объект является атомарным значением, контекстный узел не определен.

[2.0] Контекстный объект

Объект, обрабатываемый в настоящий момент. Является эквивалентом контекстного узла в Xpath 1.0; другое название отражает тот факт, что контекст может быть связан не только с узлом, но и с атомарным значением.

Позиция контекста

Положительное целое число, обозначающее позицию контекстного узла в наборе контекстных узлов. Для получения позиции контекста используется функция XPath `position()`.

Размер контекста

Положительное целое число, обозначающее количество узлов в текущем контексте. Для получения размера контекста используется функция XPath `size()`.

Привязки переменных

Набор переменных, находящихся в области видимости текущего контекста. Каждая переменная представлена именем и объектом, представляющим ее значение.

Функции

Набор всех функций, видимых в текущем контексте. Каждая функция представлена отображением между именем функции и выполняемым кодом. Каждая функция получает нуль и более аргументов и возвращает один результат. XPath определяет ряд базовых функций, которые доступны всегда; XSLT добавляет к ним дополнительные функции, выходящие за пределы спецификации XPath. Также в текущем контексте видны все функции расширения, определяемые в таблице стилей.

Объявления пространств имен

Набор объявлений пространств имен, видимых в текущем контексте. Каждое объявление состоит из префикса пространства имен и URI-адреса, с которым он связан.

[2.0] Пространство имен по умолчанию

Пространство имен по умолчанию определяется атрибутом `xpath-default-namespace`. Если в элементах, содержащих контекстный объект, такой атрибут не существует, пространство имен по умолчанию является пустым независимо от объявлений в таблице стилей (`xmlns="..."`).

[2.0] Документы и коллекции

Контекст Xpath 2.0 включает также информацию о доступных документах, доступных коллекциях и о коллекции по умолчанию. Функции `doc()`, `doc-available()` и `collection()` работают с документами и коллекциями узлов. За подробностями обращайтесь к описанию функций *[2.0]* `doc()`, *[2.0]* `doc-available()` и *[2.0]* `collection()`.

[2.0] Дополнительная информация

В Xpath 2.0 и XSLT 2.0 понятие контекста усложнилось, поэтому в контексте хранятся и другие сведения. Многие из них применимы только в определенных ситуациях (например, понятие текущей группы имеет смысл только при группировке данных), поэтому мы ограничимся их кратким перечислением:

- Текущее правило шаблона.
- Текущий режим шаблона.
- Текущая группа и текущий ключ группировки.
- Текущие сохраненные частичные совпадения (используемые с `<xsl:analyze-string>`).
- Состояние вывода, т. е. признак записи вывода в дерево результата или структуру данных. Например, в элементе `<xsl:variable>` вывод осуществляется в структуру данных.
- Неявно заданный часовой пояс.
- Набор именованных ключей, используемых функцией `key()`.
- Набор именованных числовых форматов, используемых функцией `format-number()`.

- Значения всех системных свойств, используемых функцией `system-property()`.
- Набор доступных элементов, используемых функцией `element-available()`.
- Набор всех известных схем упорядочения.
- Схема упорядочения по умолчанию.
- Базовый URI-адрес внешнего элемента; возвращается функцией XPath `base-uri()`.
- Набор определений схем в области видимости.

Типы данных XPath 1.0

Xpath 1.0 и XSLT 1.0 определяют пять типов данных, перечисленных ниже. Тип `result tree fragment` определяется XSLT 1.0 и используется только применительно к преобразованиям; другие четыре типа определяются XPath и применимы в любых технологиях, использующих XPath. Четыре типа данных XPath определяются в разделе 1 спецификации XPath; тип `result tree fragment` определяется в разделе 11.1 спецификации XSLT.

`node-set`

Набор узлов. Набор может быть как пустым, так и содержащим произвольное количество узлов.

[2.0] В XSLT 2.0 набор узлов заменяется последовательностью.

`boolean`

Представляет значение `true` или `false`. Учтите, что строки `true` и `false` не имеют специальной интерпретации или значения в XPath. Если вам потребуются логические значения, используйте функции `true()` и `false()`.

`number`

Представляет вещественное число. Все числа в XPath и XSLT реализуются как вещественные числа с плавающей точкой; тип данных `integer` (или `int`) в XPath и XSLT 1.0 не существует. Точнее говоря, все числа реализуются в стандарте вещественных чисел IEEE 754; этот же стандарт используется для примитивных типов Java `float` и `double`. Кроме обычных чисел, существуют пять специальных числовых значений: положительная и отрицательная бесконечности, положительный и отрицательный нули и `NaN` («нечисло» – специальное обозначение для всего, что не является числом).

`string`

Представляет нуль и более символов в соответствии с определением из спецификации XML.

`result tree fragment`

Временное дерево. Создается, например, элементом `<xsl:variable>`, использующим содержимое (вместо атрибута `select`) для инициализации своего значения. Значение `result tree fragment` может быть скопировано в дерево результата элементом `<xsl:copy-of>`. Возможно также его преобразование в строку элементом `<xsl:value-of>`.

[2.0] В XSLT 2.0 тип данных `result tree fragment` не существует. Дерево, сконструированное на основе входного документа, ничем не отличается от дерева, сконструированного с использованием переменной.

[2.0] Типы данных XPath 2.0

XML Schema определяет 19 примитивных типов данных; еще пять типов (`xs:anyAtomicType`, `xs:untyped`, `xs:untypedAtomic`, `xs:dayTimeDuration` и `xs:yearMonthDuration`) были добавлены в пространство имен XML Schema в спецификации XQuery 1.0 and XPath 2.0 Data Model. Все эти типы кратко представлены ниже.

В XPath 1.0 основной структурой данных является тип `node-set`; в XPath 2.0 используются последовательности. Как и `node-set`, *последовательность* может содержать типы узлов, рассмотренные ранее в этом приложении, однако она может также содержать атомарные значения типов, перечисленных ниже.

Итак, 24 типа данных XPath 2.0:

`xs:anyAtomicType`

Базовый тип для всех примитивных атомарных типов (таких как `xs:integer` или `xs:string`). Тип данных был добавлен в спецификации XQuery 1.0 and XPath 2.0.

`xs:anyURI`

Унифицированный идентификатор ресурса. Значение может быть абсолютным или относительным, а также может содержать идентификатор фрагмента. Оно должно соответствовать правилам синтаксиса URI, определенным в RFC 2396 и дополненным в RFC 2732.

`xs:base64Binary`

Произвольные двоичные данные, представленные в кодировке Base64, определяемой в RFC 2045. Допустимыми являются базовые алфавитные символы и цифры `[a-zA-Z0-9]`, а также знак «плюс» (`+`), косая черта (`/`) и знак равенства (`=`). Значение `xs:base64Binary` может также содержать произвольное количество символов-пропусков.

`xs:boolean`

Значение `true` или `false`. В XSLT и XPath строки `true` и `false` не имеют специальной интерпретации. Для получения логических значений следует использовать функции `true()` и `false()`.

`xs:date`

Дата из четырех компонентов: год, месяц, день и необязательный часовой пояс. Полное значение `xs:date` имеет формат `1995-04-21-05:00`, где `1995` – год, `04` – месяц, `21` – день, а `-05:00` – часовой пояс. Необязательный знак «минус» перед годом обозначает дату до нашей эры.

Учтите, что компоненты месяца и дня должны состоять из двух цифр, а значение года должно содержать не менее четырех цифр. Например, значение `1995-4-21` является недопустимым. Количество секунд задается с произвольной точностью.

`xs:dateTime`

Дата и время из семи компонентов: год, месяц, день, часы, минуты, секунды и необязательный часовой пояс. Полное значение `xs:dateTime` имеет формат `1995-04-21T00:05:32.6-05:00`, где `1995` – год, `04` – месяц, `21` – день, `00` – часы, `05` – минуты, `32.6` – секунды, а `-05:00` – часовой пояс. Необязательный знак «минус» перед годом обозначает дату и время до нашей эры.

Учтите, что компоненты месяца, дня, часов, минут и секунд должны состоять из двух цифр, а значение года должно содержать не менее четырех цифр. Например,

значение 1995-4-21T00:5:32.6 является недопустимым. Количество секунд задается с произвольной точностью.

xs:dayTimeDuration

Новый тип данных, определяемый в XQuery 1.0 and XPath 2.0. Является производным от xs:duration и может содержать только компоненты дней, часов, минут и секунд из xs:duration. Например, значение P2DT4H32M12.83S определяет временной промежуток продолжительностью в 2 дня, 4 часа, 32 минуты и 12.83 секунды.

xs:decimal

Последовательность цифр с точкой, отделяющей дробную часть. Значение xs:decimal может иметь начальный префикс знака (– или +). Если число не имеет дробной части, точка может отсутствовать. Примеры значений типа xs:decimal: 210, 12678967.543233, -1.23 и +100000.00.

xs:double

Число в 64-разрядном вещественном формате IEEE с двойной точностью. Значения типа xs:double, как и значения типа xs:decimal, могут иметь префикс знака (- или +). Если число не имеет дробной части, точка может отсутствовать. Значение xs:double может также содержать экспоненту: букву E или e, за которой следует показатель степени. Последний должен быть целым числом.

У типа xs:double имеются три специальных значения: INF (бесконечность), -INF (отрицательная бесконечность) и NaN («нечисло»). Примеры значений типа xs:double: -1E4, 1267.43233E12, 12.78e-2, 12, -0, 0 и INF.

xs:duration

Тип xs:duration представляет временной промежуток. Значение состоит из шести компонентов: годы, месяцы, дни, часы, минуты и секунды. Строка P2Y3M2DT7H52M23.8S определяет временной промежуток продолжительностью в 2 года, 3 месяца, 2 дня, 7 часов, 52 минуты и 23.8 секунды. Символ P обязателен. Символ T отделяет дату от времени; он может быть опущен, если ни один компонент времени не задан. Другими словами, запись P2Y3M обозначает промежуток в 2 года и 3 месяца. Компоненты, равные нулю, могут быть опущены, хотя как минимум один компонент промежутка должен быть задан. Например, запись PT2H задает промежуток в 2 часа, как и эквивалентная запись P0Y0M0DT2H0M0S.

xs:float

Число в 32-разрядном вещественном формате IEEE с одинарной точностью. Значения типа xs:float, как и значения типа xs:decimal, могут иметь префикс знака (– или +). Если число не имеет дробной части, точка может отсутствовать. Значение xs:float может также содержать экспоненту: букву E или e, за которой следует показатель степени. Последний должен быть целым числом.

У типа xs:float имеются три специальных значения: INF (бесконечность), -INF (отрицательная бесконечность) и NaN («нечисло»). Примеры значений типа xs:float: -1E4, 1267.43233E12, 12.78e-2, 12, -0, 0 и INF.

xs:gDay

День. Записывается в виде трех дефисов, за которыми следует число из двух цифр, например ---21. Поддерживается часовой пояс.

xs:gMonth

Месяц. Записывается в виде двух дефисов, за которыми следует число из двух цифр, например --04. Поддерживается часовой пояс.

`xs:gMonthDay`

День месяца. Значение `xs:gMonthDay` записывается в виде двух дефисов, месяца из двух цифр, дефиса и дня из двух цифр, например `--04-21`. Поддерживается часовой пояс.

`xs:gYear`

Год. Записывается в виде числа, содержащего не менее четырех цифр, например `1995`. Может содержать начальный знак «минус». Поддерживается часовой пояс.

`xs:gYearMonth`

Месяц года. Записывается в формате `1995-04`. Год содержит не менее четырех цифр и может иметь начальный знак «минус». Месяц должен состоять из двух цифр. Поддерживается часовой пояс.

`xs:hexBinary`

Произвольные двоичные данные, закодированные в шестнадцатеричном виде. Каждый двоичный октет в исходных данных представляется двумя шестнадцатеричными цифрами (`[0-9a-fA-F]`).

`xs:NOTATION`

Этот редко используемый тип данных представляет тип атрибута `NOTATION`, определенный в спецификации XML.

`xs:QName`

Состоит из имени пространства имен (иногда определяемого с префиксом пространства имен) и локального имени.

`xs:string`

Последовательность символов.

`xs:time`

Время из четырех компонентов: часы, минуты, секунды и необязательный часовой пояс. Полное значение `xs:time` имеет формат `00:05:32.6-05:00`, где `00` — часы, `05` — минуты, `32.6` — секунды, а `-05:00` — часовой пояс.

Значения часов, минут и секунд должны состоять из двух цифр. Количество секунд задается с произвольной точностью.

`xs:untyped`

Тип данных элемента, который не был проверен на действительность. Если элемент проверяется в режиме `skip`, то его тип также равен `xs:untyped`. Тип данных был добавлен в спецификации XQuery 1.0 and XPath 2.0.

`xs:untypedAtomic`

Нетипизованное атомарное значение. Текст, который не был связан с более конкретным типом, считается относящимся к типу `xs:untypedAtomic`, как и любой атрибут, проверяемый в режиме `skip`. Например, текст из исходного документа, не имеющего схемы, относится к типу `xs:untypedAtomic`. Тип данных был добавлен в спецификации XQuery 1.0 and XPath 2.0.

`xs:yearMonthDuration`

Новый тип данных, определяемый в XQuery 1.0 and XPath 2.0. Является производным от `xs:duration`; может содержать только компоненты года и месяца из `xs:duration`. Например, `P2Y3M` соответствует промежутку продолжительностью в два года и три месяца.

Для использования базовых типов данных не обязателен схемосовместимый процессор XSLT 2.0. Ко всем перечисленным типам данных можно применять операторы `cast as`, `castable as` и `instance of`. Однако без схемосовместимого процессора XLST невозможно определить собственные типы данных в схеме, а затем использовать их с этими операторами.

Операторы и ключевые слова

Далее приводится полный список операторов и ключевых слов XPath.

`!=` *(не равно)*

Сравнивает два операнда и возвращает `true()`, если первый операнд не равен второму. (За подробным описанием сравнения обращайтесь к разделу «Логические операторы» главы 3.)

`()`

Содержит выражение, заключенное в круглые скобки. Круглые скобки используются в математических выражениях; кроме того, в круглые скобки должны заключаться условия оператора `if`.

`*` *(квантификатор)*

Представляет нуль или более экземпляров объекта.

`*` *(умножение)*

Вычисляет произведение двух операндов. В XPath 1.0 оба операнда должны быть числами или значениями, преобразуемыми в числа.

[2.0] В XPath 2.0 значения `xs:yearMonthDuration` и `xs:dayTimeDuration` могут умножаться на числовые значения. Допустимые комбинации:

- `xs:yearMonthDuration * xs:double`
- `xs:dayTimeDuration * xs:double`

`+` *(квантификатор)*

Представляет нуль или один экземпляр объекта.

`+` *(сложение)*

Суммирует два операнда. В XPath 1.0 оба операнда должны быть числами или значениями, преобразуемыми в числа.

[2.0] В XPath 2.0 помимо числовых значений могут также суммироваться дата, время и временные промежутки. Допустимые комбинации:

- `xs:yearMonthDuration + xs:yearMonthDuration`
- `xs:dayTimeDuration + xs:dayTimeDuration`
- `xs:dateTime + xs:yearMonthDuration`
- `xs:dateTime + xs:dayTimeDuration`
- `xs:date + xs:yearMonthDuration`
- `xs:date + xs:dayTimeDuration`
- `xs:time + xs:dayTimeDuration`

`+` *(унарный плюс)*

Возвращает операнд с прежним знаком. Оператор не изменяет свой операнд.

[2.0], (оператор последовательности)

Оператор «запятая» объединяет объекты в последовательность. Например, конструкция `select="((1, 2, 3), (4, 5), 6)"` создает новую последовательность `(1, 2, 3, 4, 5, 6)`.

`-` *(унарный минус)*

Меняет знак операнда на противоположный.

`-` *(вычитание)*

Вычитает второй операнд из первого. В XPath 1.0 оба операнда должны быть числами или значениями, преобразуемыми в числа.

[2.0] В XPath 2.0 помимо числовых значений могут также суммироваться дата, время и временные промежутки. Допустимые комбинации:

- `xs:yearMonthDuration - xs:yearMonthDuration`
- `xs:dayTimeDuration - xs:dayTimeDuration`
- `xs:dateTime - xs:yearMonthDuration`
- `xs:dateTime - xs:dayTimeDuration`
- `xs:date - xs:yearMonthDuration`
- `xs:date - xs:dayTimeDuration`
- `xs:time - xs:dayTimeDuration`

`/` *(изменение уровня)*

Уровень в пути доступа.

`//` *(изменение уровня)*

Нуль и более уровней в пути доступа.

`<` *(меньше)*

Сравнивает два операнда и возвращает `true`, если первый операнд меньше второго. (За подробным описанием сравнения обращайтесь к разделу «Логические операторы» главы 3.)

[2.0] `<<` *(узел-до)*

Сравнивает два узла и возвращает `true`, если первый узел находится в документе до второго. В противном случае (в том числе и при совпадении двух узлов) возвращает `false`. Так как атрибут в XSLT не может содержать левую угловую скобку, оператор должен кодироваться в виде `<<`.

`<=` *(меньше или равно)*

Сравнивает два операнда и возвращает `true`, если первый операнд меньше второго или равен ему. (За подробным описанием сравнения обращайтесь к разделу «Логические операторы» главы 3.)

`=` *(равно)*

Сравнивает два операнда и возвращает `true`, если первый операнд равен второму. (За подробным описанием сравнения обращайтесь к разделу «Логические операторы» главы 3.)

`>` *(больше)*

Сравнивает два операнда и возвращает `true`, если первый операнд больше второго. (За подробным описанием сравнения обращайтесь к разделу «Логические операторы» главы 3.)

[2.0] >> *(узел-после)*

Сравнивает два узла и возвращает `true`, если первый узел находится в документе после второго. В противном случае (в том числе и при совпадении двух узлов) возвращает `false`.

>= *(больше или равно)*

Сравнивает два операнда и возвращает `true`, если первый операнд больше второго или равен ему. (За подробным описанием сравнения обращайтесь к разделу «Логические операторы» главы 3.)

? *(квантификатор)*

Представляет нуль или один экземпляр объекта.

[ ]

Содержит предикат в выражении XPath.

| *(объединение)*

Сравнивает два набора узлов и возвращает набор, содержащий все узлы из обоих наборов. *[2.0]* В XPath 2.0 оператор | идентичен оператору `union`.

and

Для двух заданных выражений возвращает `true`, если оба выражения истинны. Если хотя бы одно из них ложно, возвращается `false`.

[2.0] `cast as`

Преобразует значение к другому типу. Учтите, что преобразование приводит к фатальной ошибке, если значение не может быть преобразовано к новому типу. Например, конструкция `"'Lily' cast as xs:integer"` вызывает фатальную ошибку. Чтобы узнать, сработает ли преобразование, следует сначала воспользоваться оператором `castable as`.

[2.0] `castable as`

Проверяет, может ли значение быть преобразовано к другому типу. Например, выражение `"'3' castable as xs:integer"` истинно. Как нетрудно предположить, выражение `"'Lily' castable as xs:integer"` ложно. В отличие от оператора `cast as`, оператор `castable as` не выполняет преобразования. Он всего лишь сообщает, возможно данное преобразование или нет.

div *(деление)*

Делит первый операнд на второй. В XPath 1.0 оба операнда должны быть числами или значениями, преобразуемыми в числа.

[2.0] В XPath 2.0 поддерживается также деление временных промежутков:

- `xs:yearMonthDuration div xs:double`
- `xs:yearMonthDuration div xs:yearMonthDuration`
- `xs:dayTimeDuration div xs:double`
- `xs:dayTimeDuration div xs:dayTimeDuration`

[2.0] `eq` *(равно)*

Сравнивает два атомарных значения и возвращает `true`, если они равны. Оператор `eq` может использоваться для сравнения следующих *типов данных*:

- Числовые значения (`xs:decimal`, `xs:double`, `xs:float`, `xs:integer`)
- Строковые значения

- Логические значения
- Временные промежутки (`xs:duration`, `xs:yearMonthDuration`, `xs:dayTimeDuration`)
- Дата и время (`xs:date`, `xs:time`, `xs:dateTime`)
- Компоненты дат (`xs:gYear`, `xs:gYearMonth`, `xs:gMonth`, `xs:gMonthDay`, `xs:gDay`)
- Полные имена (`xs:QName`)
- Двоичные данные (`xs:hexBinary`, `xs:base64Binary`)
- Нотации (`xs:NOTATION`)

[2.0] `every`

Для заданной последовательности и условия возвращает `true`, если условие истинно для каждого объекта последовательности. Оператор записывается в форме `every $x in $sequence satisfies [test condition for $x]`.

[2.0] `except`

Сравнивает две последовательности узлов и возвращает последовательность, содержащую узлы, которые присутствуют в первой последовательности, но отсутствуют во второй. Дубликаты узлов удаляются. Оператор `except` сравнивает сами узлы, а не их значения.

[2.0] `for`

Итератор для перебора последовательности. Оператор, записанный в форме `for $x in $sequence return ...`, перебирает все значения в последовательности `$sequence`. При каждой итерации значение `$x` представляет текущее значение.

[2.0] `ge` *(больше либо равно)*

Сравнивает два атомарных значения и возвращает `true`, если первое значение больше второго или равно ему. Оператор `ge` может использоваться для сравнения следующих типов данных:

- Числовые значения (`xs:decimal`, `xs:double`, `xs:float`, `xs:integer`)
- Строковые значения
- Логические значения
- Временные промежутки (`xs:yearMonthDuration` и `xs:dayTimeDuration`, но *не* `xs:duration`)
- Дата и время (`xs:date`, `xs:time`, `xs:dateTime`)

[2.0] `gt` *(больше)*

Сравнивает два атомарных значения и возвращает `true`, если первое значение больше второго. Оператор `gt` может использоваться для сравнения следующих типов данных:

- Числовые значения (`xs:decimal`, `xs:double`, `xs:float`, `xs:integer`)
- Строковые значения
- Логические значения
- Временные промежутки (`xs:yearMonthDuration` и `xs:dayTimeDuration`, но *не* `xs:duration`)
- Дата и время (`xs:date`, `xs:time`, `xs:dateTime`)

[2.0] `idiv` *(целочисленное деление)*

Делит первый операнд на второй. Остаток игнорируется, возвращается целая часть результата.

[2.0] `if`

Вычисляет выражение, а затем выполняет одно из двух действий в зависимости от результата вычисления (`true` или `false`). Выражение должно быть заключено в круглые скобки, а оператор должен содержать секции `then` и `else`.

[2.0] `instance of`

Определяет, является ли аргумент экземпляром некоторого типа данных. Выражение `"3 instance of xs:integer"` истинно; выражение `"'j' instance of xs:integer"` ложно.

[2.0] `intersect`

Сравнивает две последовательности узлов и возвращает последовательность из узлов, присутствующих в обеих последовательностях. Дубликаты узлов удаляются. Оператор `intersect` сравнивает сами узлы, а не их значения.

[2.0] `is`

Сравнивает два узла и возвращает `true`, если это *один и тот же* узел. Если два разных узла имеют одинаковые значения, выражение `node1 is node2` возвращает `false`. Оператор `is` сравнивает сами узлы, а не их значения.

[2.0] `le` *(меньше либо равно)*

Сравнивает два атомарных значения и возвращает `true`, если первое значение меньше второго или равно ему. Оператор `le` может использоваться для сравнения следующих типов данных:

- Числовые значения (`xs:decimal`, `xs:double`, `xs:float`, `xs:integer`)
- Строковые значения
- Логические значения
- Временные промежутки (`xs:yearMonthDuration` и `xs:dayTimeDuration`, но *не* `xs:duration`)
- Дата и время (`xs:date`, `xs:time`, `xs:dateTime`)

[2.0] `lt` *(меньше)*

Сравнивает два атомарных значения и возвращает `true`, если первое значение меньше второго. Оператор `lt` может использоваться для сравнения следующих типов данных:

- Числовые значения (`xs:decimal`, `xs:double`, `xs:float`, `xs:integer`)
- Строковые значения
- Логические значения
- Временные промежутки (`xs:yearMonthDuration` и `xs:dayTimeDuration`, но *не* `xs:duration`)
- Дата и время (`xs:date`, `xs:time`, `xs:dateTime`)

`mod` *(остаток от деления)*

Делит первый операнд на второй и возвращает остаток.

[2.0] `ne` *(не равно)*

Сравнивает два атомарных значения и возвращает `true`, если они *не* равны. Оператор `ne` может использоваться для сравнения следующих типов данных:

- Числовые значения (`xs:decimal`, `xs:double`, `xs:float`, `xs:integer`)
- Логические значения

- Строковые значения
- Временные промежутки (xs:duration, xs:yearMonthDuration, xs:dayTimeDuration)
- Дата и время (xs:date, xs:time, xs:dateTime)
- Компоненты дат (xs:gYear, xs:gYearMonth, xs:gMonth, xs:gMonthDay, xs:gDay)
- Полные имена (xs:QName)
- Двоичные данные (xs:hexBinary, xs:base64Binary)
- Нотации (xs:NOTATION)

`or`

Для двух заданных выражений возвращает `true`, если хотя бы одно из двух выражений истинно. Если оба выражения ложны, возвращается `false`.

[2.0] `some`

Для заданных последовательности и условия возвращает `true`, если условие выполняется хотя бы для одного объекта последовательности. Оператор записывается в форме `some $x in $sequence satisfies [test condition for $x]`.

[2.0] `to` *(диапазон)*

Создает диапазон целых чисел. Выражение `1 to 10` создает последовательность из 10 целых чисел в порядке `1, 2, 3...`.

[2.0] `treat as`

Отключает статическую проверку типов XPath 2.0. Если переменная `$a` относится к типу `xs:integer*`, а функции должен передаваться параметр типа `xs:integer`, выражение `$a treat as xs:integer` позволит обойти статическую проверку. Если `$a` является последовательностью целых чисел, то во время выполнения `treat as` отработает без ошибок. С другой стороны, если `$a` содержит что-то другое (в том числе является пустой последовательностью), произойдет ошибка времени выполнения.

[2.0] `union`

Сравнивает две последовательности узлов и возвращает последовательность, содержащую все узлы из обеих последовательностей. Дубликаты узлов удаляются. Оператор `union` сравнивает сами узлы, а не их значения.

Приоритет операторов – XPath 1.0

Приоритеты операторов XPath 1.0, упорядоченные по возрастанию:

- `or`
- `and`
- `=`, `!=`, `<`, `<=`, `>`, `>=`
- `+` (сложение), `–` (вычитание)
- `*` (умножение), `div`, `mod`
- `|`
- `–` (унарный минус), `+` (унарный плюс)
- `/`, `//`
- `[ ]`, `( )`, `{ }`

[2.0] Приоритет операторов – XQuery 1.0 и XPath 2.0

Приоритеты операторов согласно спецификации XQuery 1.0 и XPath 2.0, упорядоченные по возрастанию:

- , (запятая)
- for, some, every, if
- or
- and
- eq, ne, lt, le, gt, ge, =, !=, <, <=, >, >=, is, <<, >>
- to
- + (сложение), – (вычитание)
- * (умножение), div, idiv, mod
- union, |
- intersect, except
- instance of
- treat as
- castable as
- cast as
- – (унарный минус), + (унарный плюс)
- ?, * (квантификатор), + (квантификатор)
- /, //
- [ ], ( ), { }

C

Справочник по функциям XSLT, XPath и XQuery

В этом разделе перечисляются все функции, определяемые в XSLT 1.0 и 2.0, XPath 1.0 и 2.0, а также в XQuery 1.0.

Классификация функций

Вместе с новыми функциями, появившимися в XSLT 2.0 и XPath 2.0, количество полезных функций превысило 100. Далее приводится список этих функций, упорядоченных по категориям.

Функции доступа

XPath 2.0 определяет функции доступа для обращения к некоторым свойствам элементов, последовательностей и документов:

- *[2.0]* `base-uri()`
- *[2.0]* `data()`
- *[2.0]* `document-uri()`
- *[2.0 – Schema]* `nilled()`
- *[2.0]* `node-name()`
- `string()`

Логические функции

Функции для работы с логическими (булевыми) значениями:

- `false()`
- `not()`
- `true()`

Конструкторы

Следующие две функции конструируют значения `xs:dateTime` и `xs:QName`:

- *[2.0]* `dateTime()`
- *[2.0]* `QName()`

Контекстные функции

Контекст XPath обладает рядом свойств. Функции данной категории предназначены для обращения к этим свойствам:

- *[2.0]* `collection()`
- `current()`
- *[2.0]* `current-date()`
- *[2.0]* `current-dateTime()`
- *[2.0]* `current-time()`
- *[2.0]* `default-collation()`
- `element-available()`
- `function-available()`
- *[2.0]* `implicit-timezone()`
- `last()`
- `position()`
- *[2.0]* `static-base-uri()`
- *[2.0]* `type-available()`

Функции `element-available()`, `function-available()` и *[2.0]* `type-available()` определяются в XSLT 2.0.

Функции перекрестных ссылок и группировки

Функции этой категории используются для разрешения уникальных идентификаторов и ссылок между элементами и атрибутами:

- *[2.0]* `current-group()`
- *[2.0]* `current-grouping-key()`
- `generate-id()`
- `id()`
- *[2.0]* `idref()`
- `key()`

Функции даты, времени и временных промежутков

Функции этой категории предназначены для работы с датами, временем и временными интервалами. В XML Schema определены типы `xs:date`, `xs:dateTime`, `xs:dayTimeDuration`, `xs:duration`, `xs:time` и `xs:yearMonthDuration`:

- *[2.0]* `adjust-date-to-timezone()`
- *[2.0]* `adjust-dateTime-to-timezone()`

- *[2.0]* adjust-time-to-timezone()
- *[2.0]* day-from-date()
- *[2.0]* day-from-dateTime()
- *[2.0]* days-from-duration()
- *[2.0]* format-date()
- *[2.0]* format-dateTime()
- *[2.0]* format-time()
- *[2.0]* hours-from-dateTime()
- *[2.0]* hours-from-duration()
- *[2.0]* hours-from-time()
- *[2.0]* minutes-from-dateTime()
- *[2.0]* minutes-from-duration()
- *[2.0]* minutes-from-time()
- *[2.0]* month-from-date()
- *[2.0]* month-from-dateTime()
- *[2.0]* months-from-duration()
- *[2.0]* seconds-from-dateTime()
- *[2.0]* seconds-from-duration()
- *[2.0]* seconds-from-time()
- *[2.0]* timezone-from-date()
- *[2.0]* timezone-from-dateTime()
- *[2.0]* timezone-from-time()
- *[2.0]* year-from-date()
- *[2.0]* year-from-dateTime()
- *[2.0]* years-from-duration()

Функции *[2.0]* format-date(), *[2.0]* format-dateTime() и *[2.0]* format-time() определяются в XSLT 2.0.

Функции узлов

Функции этой категории выполняют операции с узлами:

- lang()
- local-name()
- name()
- namespace-uri()
- number()
- *[2.0]* root()

Числовые функции

Функции для работы с числовыми значениями. Некоторые работают с одним числом, остальные – с наборами узлов или последовательностями:

- *[2.0]* `abs()`
- *[2.0]* `avg()`
- `ceiling()`
- `floor()`
- `format-number()`
- *[2.0]* `max()`
- *[2.0]* `min()`
- `round()`
- *[2.0]* `round-half-to-even()`
- `sum()`

Функции QName

Функции этой категории предназначены для работы с QName (тип данных XML Schema `xs:QName`):

- *[2.0]* `in-scope-prefixes()`
- *[2.0]* `local-name-from-QName()`
- *[2.0]* `namespace-uri-for-prefix()`
- *[2.0]* `namespace-uri-from-QName()`
- *[2.0]* `prefix-from-QName()`
- *[2.0]* `QName()`
- *[2.0]* `resolve-QName()`

Функции регулярных выражений

Эти функции являются частью поддержки регулярных выражений в XSLT 2.0 и XPath 2.0:

- *[2.0]* `matches()`
- *[2.0]* `regex-group()`
- *[2.0]* `replace()`
- *[2.0]* `tokenize()`

Функции последовательностей и наборов узлов

Функции этой категории работают с *[1.0]* наборами узлов или *[2.0]* последовательностями:

- *[2.0]* `avg()`
- `boolean()`
- `count()`
- *[2.0]* `deep-equal()`
- *[2.0]* `distinct-values()`
- *[2.0]* `doc-available()`
- *[2.0]* `doc()`
- *[2.0]* `empty()`

- *[2.0]* exactly-one()
- *[2.0]* exists()
- *[2.0]* index-of()
- *[2.0]* insert-before()
- *[2.0]* max()
- *[2.0]* min()
- *[2.0]* one-or-more()
- *[2.0]* remove()
- *[2.0]* reverse()
- *[2.0]* subsequence()
- sum()
- *[2.0]* unordered()
- *[2.0]* zero-or-one()

Строковые функции

Функции этой категории используются при работе со строками:

- *[2.0]* codepoint-equal()
- *[2.0]* codepoints-to-string()
- *[2.0]* compare()
- concat()
- contains()
- *[2.0]* encode-for-uri()
- *[2.0]* ends-with()
- *[2.0]* escape-html-uri()
- *[2.0]* iri-to-uri()
- *[2.0]* lower-case()
- *[2.0]* matches()
- normalize-space()
- *[2.0]* normalize-unicode()
- *[2.0]* replace()
- starts-with()
- *[2.0]* string-join()
- string-length()
- *[2.0]* string-to-codepoints()
- substring-after()
- substring-before()
- substring()
- *[2.0]* tokenize()
- translate()
- *[2.0]* upper-case()

Прочие функции

Здесь собраны все функции, не относящиеся ни к какой другой категории.

- *[2.0]* `collection()`
- *[2.0]* `doc-available()`
- *[2.0]* `doc()`
- `document()`
- *[2.0]* `error()`
- *[2.0]* `resolve-uri()`
- `system-property()`
- *[2.0]* `trace()`
- *[2.0]* `unparsed-entity-public-id()`
- `unparsed-entity-uri()`
- *[2.0]* `unparsed-text-available()`
- *[2.0]* `unparsed-text()`

Функции упорядочения

В эту категорию объединены функции, используемые для задания схем упорядочения, обеспечивающих сравнение или сортировку текстов по правилам конкретных языков. (Все эти функции уже были перечислены ранее в составе строковых функций или функций последовательностей. Приведенный ниже список содержит все функции, работающие со схемой упорядочения.)

- *[2.0]* `compare()`
- `contains()`
- *[2.0]* `deep-equal()`
- *[2.0]* `distinct-values()`
- *[2.0]* `index-of()`
- `substring-after()`
- `substring-before()`

[2.0] abs()

Возвращает модуль (абсолютное значение) числового аргумента.

Синтаксис

```
numeric? abs(numeric?)
```

Ввод

Числовое значение.

Вывод

Модуль заданного числового значения.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 6.4 «Functions on Numeric Values».

Пример

Короткая таблица стилей, демонстрирующая применение функции `abs()`:

```
<?xml version="1.0"?>
<!-- abs.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Here are some tests of the abs() function:&#xA;</xsl:text>
    <xsl:text>&#xA;  abs(7) = </xsl:text>
    <xsl:value-of select="abs(7)"/>
    <xsl:text>&#xA;  abs(-7) = </xsl:text>
    <xsl:value-of select="abs(-7)"/>
    <xsl:text>&#xA;  abs(0) = </xsl:text>
    <xsl:value-of select="abs(0)"/>
    <xsl:text>&#xA;  abs(-0) = </xsl:text>
    <xsl:value-of select="abs(-0)"/>

    <!-- В процессоре XSLT 2.0 этот пример работать не будет. -->
    <!-- <xsl:value-of select="abs('x')"/> -->

    <xsl:variable name="testSequence" as="xs:integer*" select="1 to 10"/>
    <xsl:text>&#xA;  $testSequence = </xsl:text>
    <xsl:value-of select="$testSequence" separator=", "/>
    <xsl:text>&#xA;  abs(count($testSequence)) = </xsl:text>
    <xsl:value-of select="abs(count($testSequence))"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here are some tests of the abs() function:

  abs(7) = 7
  abs(-7) = 7
  abs(0) = 0
  abs(-0) = 0
  $testSequence = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
  abs(count($testSequence)) = 10
```

В последнем вызове `abs()` функция `count()` используется для создания числового значения. Как сказано в комментарии в таблице стилей, любая попытка передачи нечислового значения функции `abs()` является статической ошибкой. Если хотите, раскомментируйте строку `<xsl:value-of select="abs('x')/>` и убедитесь в этом сами.

[2.0] adjust-date-to-timezone()

Изменяет значение `xs:date` для заданного часового пояса.

Синтаксис

```
xs:date? adjust-date-to-timezone(xs:date?)
xs:date? adjust-date-to-timezone(xs:date?, $timezone as xs:dayTimeDuration)
```

Ввод

Необязательное значение `xs:date` и необязательное значение `xs:dayTimeDuration`. Если значение `xs:date` не задано, возвращается пустая последовательность. Если значение `xs:dayTimeDuration` задано, дата адаптируется для содержащегося в нем часового пояса; в противном случае процессор XSLT использует часовой пояс по умолчанию, возвращаемый функцией *[2.0]* `implicit-timezone()`. Наконец, если часовой пояс, определяемый `xs:dayTimeDuration`, является пустой последовательностью, функция возвращает дату с удаленной информацией часового пояса.

Вывод

Значение `xs:date`, адаптированное для заданного часового пояса.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.7 «Timezone Adjustment Functions on Dates and Time Values».

Пример

Таблица стилей, демонстрирующая различные варианты использования функции `adjust-date-to-timezone()`:

```
<?xml version="1.0"?>
<!-- adjust-date-to-timezone.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="gmt" select="xs:dayTimeDuration('PT0H')"/>
    <xsl:variable name="est" select="xs:dayTimeDuration('-PT5H')"/>
    <xsl:variable name="cst" select="xs:dayTimeDuration('-PT6H')"/>
    <xsl:variable name="minusTen" select="xs:dayTimeDuration('-PT10H')"/>

    <xsl:variable name="LilysBirthday" as="xs:date"
      select="xs:date('1995-04-21')"/>
    <xsl:variable name="format"
      select="'[FNn], [MNn] [D1], [Y0001]'"/>

    <xsl:text>&#xA;Here are some tests of the </xsl:text>
    <xsl:text>adjust-date-to-timezone() function:&#xA;</xsl:text>

    <xsl:text>&#xA;  My daughter was born on &#x9;</xsl:text>
    <xsl:value-of
      select="format-date($LilysBirthday, $format)"/>
    <xsl:text>&#xA;    adjusted to GMT: &#x9;&#x9;</xsl:text>
    <xsl:value-of
```

```
        select="format-date(
                adjust-date-to-timezone($LilysBirthday, $gmt),
                $format)"/>

      <xsl:text>&#xA;    adjusted to EST: &#x9;&#x9;</xsl:text>
      <xsl:value-of
        select="format-date
                (adjust-date-to-timezone($LilysBirthday, $est),
                $format)"/>

      <xsl:text>&#xA;    adjusted to CST: &#x9;&#x9;</xsl:text>
      <xsl:value-of
        select="format-date
                (adjust-date-to-timezone($LilysBirthday, $cst),
                $format)"/>

      <xsl:text>&#xA;&#xA;  </xsl:text>
      <xsl:text>The current date in the default timezone is: </xsl:text>
      <xsl:text>&#xA;&#x9;&#x9;&#x9;&#x9;</xsl:text>
      <xsl:value-of
        select="format-date(
                adjust-date-to-timezone(current-date()),
                $format)"/>

      <xsl:text>&#xA;    adjusted to GMT: &#x9;&#x9;</xsl:text>
      <xsl:value-of
        select="format-date
                (adjust-date-to-timezone(current-date(), $gmt),
                $format)"/>

      <xsl:text>&#xA;    adjusted to GMT-10: &#x9;</xsl:text>
      <xsl:value-of
        select="format-date(
                adjust-date-to-timezone(current-date(), $minusTen),
                $format)"/>
    </xsl:template>

  </xsl:stylesheet>
```

Результат:

```
  Here are some tests of the adjust-date-to-timezone() function:

    My daughter was born on       Friday, April 21, 1995
      adjusted to GMT:            Friday, April 21, 1995
      adjusted to EST:            Friday, April 21, 1995
      adjusted to CST:            Friday, April 21, 1995

    The current date in the default timezone is:
                                  Thursday, November 16, 2007
      adjusted to GMT:            Thursday, November 16, 2007
      adjusted to GMT-10:         Wednesday, November 15, 2007
```

Обратите внимание на использование функции `format-date()` **для вывода значений** `xs:date`**.**

См. также

Определения функций ***[2.0]*** `adjust-dateTime-to-timezone()`**,** ***[2.0]*** `adjust-time-to-timezone()` **и** ***[2.0]*** `format-date()`**.**

[2.0] adjust-dateTime-to-timezone()

Изменяет значение xs:dateTime для заданного часового пояса.

Синтаксис

```
xs:dateTime? adjust-dateTime-to-timezone(xs:dateTime?)
xs:dateTime? adjust-dateTime-to-timezone(xs:dateTime?, $timezone as xs:dayTimeDuration?)
```

Ввод

Необязательное значение xs:dateTime и необязательное значение xs:dayTimeDuration. Если значение xs:dateTime не задано, возвращается пустая последовательность. Если значение xs:dayTimeDuration задано, xs:dateTime адаптируется для содержащегося в нем часового пояса. В противном случае adjust-dateTime-to-timezone() адаптирует xs:dateTime для часового пояса по умолчанию, возвращаемого функцией *[2.0]* implicit-timezone(). Наконец, если заданный часовой пояс является пустой последовательностью, функция возвращает xs:dateTime с удаленной информацией часового пояса.

Вывод

Значение xs:dateTime, адаптированное для заданного часового пояса.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.7 «Timezone Adjustment Functions on Dates and Time Values».

Пример

Таблица стилей демонстрирует различные варианты использования функции adjust-dateTime-to-timezone(). По умолчанию используется часовой пояс GMT-4, соответствующий восточному летнему времени (EDT, Eastern Daylight Time).

```
<?xml version="1.0"?>
<!-- adjust-datetime-to-timezone.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="gmt" select="xs:dayTimeDuration('PT0H')"/>
    <xsl:variable name="est" select="xs:dayTimeDuration('-PT5H')"/>
    <xsl:variable name="cst" select="xs:dayTimeDuration('-PT6H')"/>
    <xsl:variable name="minusTen" select="xs:dayTimeDuration('-PT10H')"/>

    <xsl:variable name="LilysBirthday" as="xs:dateTime"
      select="xs:dateTime('1995-04-21T00:43:00-05:00')"/>
    <xsl:variable name="format"
      select="'[FNn], [MNn] [D1], [Y0001], at [h1]:[m01]:[s01]'"/>

    <xsl:text>&#xA;Here are some tests of the </xsl:text>
    <xsl:text>adjust-dateTime-to-timezone() function:&#xA;</xsl:text>

    <xsl:text>&#xA;  My daughter was born on &#x9;</xsl:text>
    <xsl:value-of
      select="format-dateTime($LilysBirthday, $format)"/>
```

```
    <xsl:text>&#xA;    adjusted to GMT: &#x9;&#x9;</xsl:text>
    <xsl:value-of
      select="format-dateTime(
              adjust-dateTime-to-timezone($LilysBirthday, $gmt),
              $format)"/>

    <xsl:text>&#xA;    adjusted to EST: &#x9;&#x9;</xsl:text>
    <xsl:value-of
      select="format-dateTime
              (adjust-dateTime-to-timezone($LilysBirthday, $est),
              $format)"/>

    <xsl:text>&#xA;    adjusted to CST: &#x9;&#x9;</xsl:text>
    <xsl:value-of
      select="format-dateTime
              (adjust-dateTime-to-timezone($LilysBirthday, $cst),
              $format)"/>

    <xsl:text>&#xA;&#xA;  </xsl:text>
    <xsl:text>The current time in the default timezone is: </xsl:text>
    <xsl:text>&#xA;  &#x9;&#x9;&#x9;&#x9;</xsl:text>
    <xsl:value-of
      select="format-dateTime(
              adjust-dateTime-to-timezone(current-dateTime()),
              $format)"/>

    <xsl:text>&#xA;    adjusted to GMT: &#x9;&#x9;</xsl:text>
    <xsl:value-of
      select="format-dateTime
              (adjust-dateTime-to-timezone(current-dateTime(), $gmt),
              $format)"/>

    <xsl:text>&#xA;    adjusted to GMT-10: &#x9;</xsl:text>
    <xsl:value-of
      select="format-dateTime(
              adjust-dateTime-to-timezone(current-dateTime(), $minusTen),
              $format)"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here are some tests of the adjust-dateTime-to-timezone() function:

  My daughter was born on       Friday, April 21, 1995, at 12:43:00
    adjusted to GMT:            Friday, April 21, 1995, at 5:43:00
    adjusted to EST:            Friday, April 21, 1995, at 12:43:00
    adjusted to CST:            Thursday, April 20, 1995, at 11:43:00

  The current time in the default timezone is:
                                Thursday, November 16, 2006, at 3:34:29
    adjusted to GMT:            Thursday, November 16, 2006, at 8:34:29
    adjusted to GMT-10:         Wednesday, November 15, 2006, at 10:34:29
```

Часовой пояс по умолчанию считается частью контекста; значение по умолчанию может быть получено функцией `implicit-timezone()`**. За дополнительной информацией обращайтесь к определению функции *[2.0]*** `implicit-timezone()`**. Также обратите внимание на использование функции** `format-dateTime()` **для форматирования значений** `xs:dateTime`**.**

См. также

Определения функций *[2.0]* `adjust-date-to-timezone()`, *[2.0]* `adjust-time-to-timezone()` и *[2.0]* `format-dateTime()`.

[2.0] adjust-time-to-timezone()

Изменяет значение `xs:time` для заданного часового пояса.

Синтаксис

```
xs:time? adjust-time-to-timezone(xs:time?)
xs:time? adjust-time-to-timezone(xs:time?, $timezone as xs:dayTimeDuration?)
```

Ввод

Необязательное значение `xs:time` и необязательное значение `xs:dayTimeDuration`. Если значение `xs:time` не задано, возвращается пустая последовательность. Если значение `xs:dayTimeDuration` задано, `xs:time` адаптируется для содержащегося в нем часового пояса; в противном случае процессор XSLT использует часовой пояс по умолчанию, возвращаемый функцией *[2.0]* `implicit-timezone()`. Наконец, если часовой пояс, определяемый `xs:dayTimeDuration`, является пустой последовательностью, функция возвращает `xs:time` с удаленной информацией часового пояса.

Вывод

Значение `xs:time`, адаптированное для заданного часового пояса.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.7 «Timezone Adjustment Functions on Dates and Time Values».

Пример

Таблица стилей для тестирования функции `adjust-time-to-timezone()`:

```
<?xml version="1.0"?>
<!-- adjust-time-to-timezone.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="gmt" select="xs:dayTimeDuration('PT0H')"/>
    <xsl:variable name="est" select="xs:dayTimeDuration('-PT5H')"/>
    <xsl:variable name="cst" select="xs:dayTimeDuration('-PT6H')"/>
    <xsl:variable name="minusTen" select="xs:dayTimeDuration('-PT10H')"/>

    <xsl:variable name="LilysBirthday" as="xs:time"
      select="xs:time('00:43:00-05:00')"/>
    <xsl:variable name="format"
      select="'[h1]:[m01]:[s01] [P]'"/>

    <xsl:text>&#xA;Here are some tests of the </xsl:text>
    <xsl:text>adjust-time-to-timezone() function:&#xA;</xsl:text>
```

```
    <xsl:text>&#xA;  My daughter was born at &#x9;</xsl:text>
    <xsl:value-of
      select="format-time($LilysBirthday, $format)"/>
    <xsl:text>&#xA;    adjusted to GMT: &#x9;&#x9;</xsl:text>
    <xsl:value-of
      select="format-time(
              adjust-time-to-timezone($LilysBirthday, $gmt),
              $format)"/>

    <xsl:text>&#xA;    adjusted to EST: &#x9;&#x9;</xsl:text>
    <xsl:value-of
      select="format-time
              (adjust-time-to-timezone($LilysBirthday, $est),
              $format)"/>

    <xsl:text>&#xA;    adjusted to CST: &#x9;&#x9;</xsl:text>
    <xsl:value-of
      select="format-time
              (adjust-time-to-timezone($LilysBirthday, $cst),
              $format)"/>

    <xsl:text>&#xA;&#xA;  </xsl:text>
    <xsl:text>The current time in the default timezone is: </xsl:text>
    <xsl:text>&#xA;  &#x9;&#x9;&#x9;&#x9;</xsl:text>
    <xsl:value-of
      select="format-time(
              adjust-time-to-timezone(current-time()),
              $format)"/>

    <xsl:text>&#xA;    adjusted to GMT: &#x9;&#x9;</xsl:text>
    <xsl:value-of
      select="format-time
              (adjust-time-to-timezone(current-time(), $gmt),
              $format)"/>

    <xsl:text>&#xA;    adjusted to GMT-10: &#x9;</xsl:text>
    <xsl:value-of
      select="format-time(
              adjust-time-to-timezone(current-time(), $minusTen),
              $format)"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here are some tests of the adjust-time-to-timezone() function:

  My daughter was born at       12:43:00 a.m.
    adjusted to GMT:            5:43:00 a.m.
    adjusted to EST:            12:43:00 a.m.
    adjusted to CST:            11:43:00 p.m.

  The current time in the default timezone is:
                                3:35:44 a.m.
    adjusted to GMT:            8:35:44 a.m.
    adjusted to GMT-10:         10:35:44 p.m.
```

Обратите внимание на использование функции `format-time()` **для вывода значений** `xs:time`**.**

См. также

Определения функций *[2.0]* `adjust-date-to-timezone()`, *[2.0]* `adjust-dateTime-to-timezone()` и *[2.0]* `format-time()`.

[2.0] avg()

Возвращает для заданной последовательности среднее арифметическое всех значений.

Синтаксис

```
xs:anyAtomicType? avg(xs:anyAtomicType*)
```

Ввод

Последовательность значений.

Вывод

Среднее арифметическое значений заданной последовательности. Среднее значение может вычисляться для шести типов данных: `xs:integer`, `xs:double`, `xs:decimal`, `xs:float`, `xs:yearMonthDuration` и `xs:dayTimeDuration`.

Для последовательности числовых значений процессор XSLT возвращает среднее арифметическое чисел, преобразуя типы данных по мере необходимости. Для последовательности временных промежутков процессор XSLT возвращает среднюю величину промежутка.

Функция `avg()` предполагает, что переданная последовательность содержит осмысленные данные; в противном случае процессор XSLT выдает ошибку. Например, при вызове для последовательности `(42, 57, 'blue')` процессор XSLT, как и следовало ожидать, вернет ошибку.

Несколько замечаний по поводу работы функции `avg()`:

- Для вычисления среднего значения для последовательности временных промежутков все значения должны относиться к типу `xs:dayTimeDurations` или `xs:yearMonthDurations`. Смешанное использование двух типов недопустимо; если вы попытаетесь это сделать, процессор XSLT выдаст ошибку.
- Если все значения последовательности относятся к типу `xs:untypedAtomic`, процессор XSLT пытается преобразовать каждое значение к типу `xs:double`. Если *хотя бы одно* значение в последовательности не удается преобразовать к `xs:double`, процессор XSLT выдает ошибку.
- Наконец, если передать `avg()` пустую последовательность, функция вернет пустую последовательность. Хотя в этом случае функция не получает никаких полезных данных, процессор XSLT не выдает ошибку.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.4 «Aggregate Functions».

Пример

Таблица стилей для вычисления средних значений нескольких последовательностей:

```
<?xml version="1.0"?>
<!-- avg.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="seq1" select="(3, 5, 18)"/>
    <xsl:variable name="seq2" select="(3, 5, 48.273, 2.9e3)"/>

    <xsl:variable name="value1" as="xs:integer" select="42"/>
    <xsl:variable name="value2" as="xs:double" select="2718.28E-3"/>
    <xsl:variable name="value3" as="xs:float" select="98.6"/>
    <xsl:variable name="value4" as="xs:decimal" select="2.54"/>
    <xsl:variable name="seq3"
      select="($value1, $value2, $value3, $value4)"/>

    <xsl:variable name="seq4"
      select="(xs:yearMonthDuration('P3Y8M'),
               xs:yearMonthDuration('P4Y2M'),
               xs:yearMonthDuration('P6Y4M'))"/>
    <xsl:variable name="seq5"
      select="(xs:dayTimeDuration('P2DT4H23M12.2S'),
               xs:dayTimeDuration('P3DT8H17M'),
               xs:dayTimeDuration('P3D'))"/>

    <xsl:text>&#xA;Here are some tests of the avg() function:&#xA;</xsl:text>

    <xsl:text>&#xA;  avg(</xsl:text>
    <xsl:value-of select="$seq1" separator=", "/>
    <xsl:text>) = </xsl:text>
    <xsl:value-of select="format-number(avg($seq1), '#.###')"/>

    <xsl:text>&#xA;&#xA;  avg(</xsl:text>
    <xsl:value-of select="$seq2" separator=", "/>
    <xsl:text>) = </xsl:text>
    <xsl:value-of select="format-number(avg($seq2), '#.###')"/>

    <xsl:text>&#xA;&#xA;  avg(</xsl:text>
    <xsl:value-of select="$seq3" separator=", "/>
    <xsl:text>) = </xsl:text>
    <xsl:value-of select="format-number(avg($seq3), '#.###')"/>

    <xsl:text>&#xA;&#xA;  avg(</xsl:text>
    <xsl:value-of select="$seq4" separator=", "/>
    <xsl:text>) = </xsl:text>
    <xsl:value-of select="avg($seq4)"/>

    <xsl:text>&#xA;&#xA;    In text, the average of</xsl:text>
    <xsl:for-each select="$seq4">
      <xsl:text>&#xA;      </xsl:text>
      <xsl:value-of select="years-from-duration(.)"/>
      <xsl:text> years and </xsl:text>
      <xsl:value-of select="months-from-duration(.)"/>
      <xsl:text> months (</xsl:text>
      <xsl:value-of select="."/>
      <xsl:text>)</xsl:text>
    </xsl:for-each>
```

```
    <xsl:text>&#xA;    is </xsl:text>
    <xsl:value-of select="years-from-duration(avg($seq4))"/>
    <xsl:text> years and </xsl:text>
    <xsl:value-of select="months-from-duration(avg($seq4))"/>
    <xsl:text> months (</xsl:text>
    <xsl:value-of select="avg($seq4)"/>
    <xsl:text>).</xsl:text>

    <xsl:text>&#xA;&#xA;  avg(</xsl:text>
    <xsl:value-of select="$seq5" separator=", "/>
    <xsl:text>) = </xsl:text>
    <xsl:variable name="avg5" select="avg($seq5)"/>
    <xsl:value-of select="$avg5"/>

    <xsl:text>&#xA;&#xA;    In text, the average of</xsl:text>
    <xsl:for-each select="$seq5">
      <xsl:text>&#xA;      </xsl:text>
      <xsl:value-of select="days-from-duration(.)"/>
      <xsl:text> days, </xsl:text>
      <xsl:value-of select="hours-from-duration(.)"/>
      <xsl:text> hours, </xsl:text>
      <xsl:value-of select="minutes-from-duration(.)"/>
      <xsl:text> minutes and </xsl:text>
      <xsl:value-of
        select="format-number(seconds-from-duration(.), '#.##')"/>
      <xsl:text> seconds (</xsl:text>
      <xsl:value-of select="."/>
      <xsl:text>)</xsl:text>
    </xsl:for-each>

    <xsl:text>&#xA;    is </xsl:text>
      <xsl:value-of select="days-from-duration($avg5)"/>
      <xsl:text> days, </xsl:text>
      <xsl:value-of select="hours-from-duration($avg5)"/>
      <xsl:text> hours, </xsl:text>
      <xsl:value-of select="minutes-from-duration($avg5)"/>
      <xsl:text> minutes and </xsl:text>
      <xsl:value-of
        select="format-number(seconds-from-duration($avg5), '#.##')"/>
      <xsl:text> seconds.</xsl:text>

  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here are some tests of the avg() function:

  avg(3, 5, 18) = 8.667

  avg(3, 5, 48.273, 2900) = 739.068

  avg(42, 2.71828, 98.6, 2.54) = 36.465

  avg(P3Y8M, P4Y2M, P6Y4M) = P4Y9M

    In text, the average of
      3 years and 8 months (P3Y8M)
      4 years and 2 months (P4Y2M)
      6 years and 4 months (P6Y4M)
    is 4 years and 9 months (P4Y9M).
```

```
avg(P2DT4H23M12.2S, P3DT8H17M, P3D) = P2DT20H13M24.066666S

  In text, the average of
    2 days, 4 hours, 23 minutes and 12.2 seconds (P2DT4H23M12.2S)
    3 days, 8 hours, 17 minutes and 0 seconds (P3DT8H17M)
    3 days, 0 hours, 0 minutes and 0 seconds (P3D)
  is 2 days, 20 hours, 13 minutes and 24.07 seconds.
```

Таблица стилей демонстрирует пять разных типов последовательностей. Первая последовательность состоит только из целых чисел, вторая содержит произвольные числа. Все значения первых двух последовательностей могут быть преобразованы в числа, поэтому результат оказывается таким, каким мы и ожидали. Третья последовательность состоит из четырех числовых значений, относящихся к разным числовым типам (`xs:integer`, `xs:decimal`, `xs:float` и `xs:double`). Последние две последовательности состоят из двух типов временных промежутков. Кроме того, в таблице стилей для форматирования данных используются такие функции, как `years-from-duration()` и `format-number()`.

С последовательностью `(3, 5, '18')` функция не работает, потому что одно из значений последовательности представляет собой строку. Хотя строка `'18'` может быть преобразована в число функцией `number()`, функция `avg()` возлагает эту обязанность на нас: она требует, чтобы все литералы в последовательности были числами или временными промежутками. Впрочем, если последовательность содержит нетипизованные данные (например, узлы, прочитанные из документа, не использующего схему), процессор XSLT пытается преобразовать все эти значения к типу `xs:double`. Если хотя бы одно из них не удается преобразовать к типу `xs:double`, процессор XSLT выдает ошибку.

Последнее замечание: если вы хотите сохранить вычисленное среднее значение в типизованной переменной, тип данных `xs:double` обеспечивает наибольшую гибкость. Использовать самый ограниченный числовой тип `xs:integer` определенно не рекомендуется. Например, следующий элемент процессор XSLT обработает без всяких возражений:

```
<xsl:variable name="avg1" as="xs:integer" select="avg((3, 5, 4))"/>
```

А в этом случае будет выдана статическая ошибка:

```
<xsl:variable name="avg2" as="xs:integer" select="avg((3, 5, 18))"/>
```

Если функция `avg()` используется для динамического задания значения целочисленной переменной, процессор XSLT выдаст ошибку в том случае, если возвращаемое значение не относится к типу `xs:integer`. Чтобы вызов функции работал, результат необходимо преобразовать к `xs:integer`:

```
<xsl:variable name="avg2" as="xs:integer"
  select="xs:integer(avg((3, 5, 18)))"/>
```

[2.0] base-uri()

Возвращает базовый URI-адрес для заданного узла.

Синтаксис

```
xs:anyURI? base-uri(node()?)
xs:anyURI? base-uri()
```

Ввод

Узел. Без аргумента `base-uri()` возвращает базовый URI-адрес контекстного объекта.

Вывод

Базовый URI-адрес заданного узла.

Функция `base-uri()` работает в соответствии со спецификацией XML Base. По умолчанию базовый URI-адрес является URI-адресом самого документа XML, но документ может изменить это значение при помощи атрибута `xml:base` любого элемента.

Если заданный узел не имеет свойства базового URI-адреса, но имеет родителя, `base-uri()` просматривает его предков. Функция пытается найти базовый URI-адрес родителя узла, потом родителя его родителя и т. д., пока не найдет свойство с базовым URI-адресом или не достигнет узла, не имеющего родителя. Если свойство найти так и не удается, `base-uri()` возвращает пустую последовательность. Если аргумент `base-uri()` является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 2 «Accessors».

Пример

В качестве исходного XML-документа будет использоваться видоизмененная версия списка автомашин. Обратите внимание: все элементы `<manufacturer>` используют атрибут `xml:base` для определения нового базового URI-адреса:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- xmlbase.xml -->
<cars>
  <manufacturer name="Chevrolet"
    xml:base="http://www.chevrolet.com/">
    <car>Cavalier</car>
    <car>Corvette</car>
    <car>Impala</car>
    <car>Malibu</car>
  </manufacturer>
  <manufacturer name="Ford"
    xml:base="http://www.ford.com/">
    <car>Pinto</car>
    <car>Mustang</car>
    <car>Taurus</car>
  </manufacturer>
  <manufacturer name="Volkswagen"
    xml:base="http://www.vw.com/">
    <car>Beetle</car>
    <car>Jetta</car>
    <car>Passat</car>
    <car>Touraeg</car>
  </manufacturer>
</cars>
```

Для тестирования функции `base-uri()` будет использоваться следующая таблица стилей:

```
<?xml version="1.0"?>
<!-- base-uri.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the base-uri() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  The base URI for the </xsl:text>
    <xsl:text>document root is:&#xA;    </xsl:text>
    <xsl:value-of select="base-uri()"/>

    <xsl:text>&#xA;&#xA;  The base URI for the </xsl:text>
    <xsl:text>&lt;cars&gt; element is:&#xA;    </xsl:text>
    <xsl:value-of select="base-uri(cars)"/>

    <xsl:for-each select="/cars/manufacturer">
      <xsl:text>&#xA;&#xA;  The base URI for the </xsl:text>
      <xsl:text>manufacturer named </xsl:text>
      <xsl:value-of select="@name"/>
      <xsl:text>: &#xA;    </xsl:text>
      <xsl:value-of select="base-uri()"/>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Применение этой таблицы стилей к измененному списку автомашин дает следующий результат:

```
Tests of the base-uri() function:

  The base URI for the document root is:
    file:/C:/projects/XSLTbookV2/AppendixC/xmlbase.xml

  The base URI for the <cars> element is:
    file:/C:/projects/XSLTbookV2/AppendixC/xmlbase.xml

  The base URI for the manufacturer named Chevrolet:
    http://www.chevrolet.com/

  The base URI for the manufacturer named Ford:
    http://www.ford.com/

  The base URI for the manufacturer named Volkswagen:
    http://www.vw.com/
```

Функция `base-uri()` **особенно полезна при построении ссылок на основе документа XML. Следующая таблица стилей создает ссылки для элементов** `<car>` **с использованием функции** `base-uri()`**:**

```
<?xml version="1.0"?>
<!-- base-uri2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" include-content-type="no"/>

  <xsl:template match="/">
    <html>
      <head>
```

```
        <title>Here are some cars</title>
      </head>
      <body>
        <h1>Here are some cars:</h1>
        <xsl:apply-templates select="cars/manufacturer"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="manufacturer">
    <h2>
      <xsl:value-of select="@name"/>
    </h2>
    <ul>
      <xsl:for-each select="car">
        <li>
          <a href="{concat(base-uri(), ., '.html')}">
            <xsl:value-of select="."/>
          </a>
        </li>
      </xsl:for-each>
    </ul>
  </xsl:template>

</xsl:stylesheet>
```

При построении ссылок используются шаблон значения атрибута и функция `concat()`, объединяющая значение базового URI-адреса, значение текущего элемента (точка представляет элемент `<car>`, обрабатываемый в настоящий момент) и строку `.html`. Сгенерированная серия ссылок выглядит примерно так:

```
<h2>Volkswagen</h2>
<ul>
   <li><a href="http://www.vw.com/Beetle.html">Beetle</a></li>
   <li><a href="http://www.vw.com/Jetta.html">Jetta</a></li>
   <li><a href="http://www.vw.com/Passat.html">Passat</a></li>
   <li><a href="http://www.vw.com/Touraeg.html">Touraeg</a></li>
</ul>
```

Предполагается, что свойство базового URI-адреса завершается косой чертой, как во всех атрибутах `xml:base` исходного документа. Вероятно, для повышения надежности кода следовало бы проверить возвращаемое значение функции `base-uri()` и при необходимости присоединить к нему косую черту.

boolean()

Преобразует аргумент в логическое значение.

Синтаксис

```
[1.0] boolean boolean(object)
[2.0] xs:boolean boolean(item()*)
```

Ввод

Объект, преобразуемый в логическое значение. Процедура преобразования описана в следующем подразделе.

Вывод

[1.0] Логическое значение, соответствующее входному объекту. Объекты преобразуются в логические значения по следующим правилам:

- Число преобразуется в `true` в том и только в том случае, если оно отлично от нуля, отрицательного нуля или `NaN` («нечисло»).
- Набор узлов преобразуется в `true` в том и только в том случае, если он не пуст.
- Строка преобразуется в `true` в том и только в том случае, если она имеет положительную длину.
- Все остальные типы данных преобразуются по правилам, специфическим для этих типов.

[2.0] В XSLT 2.0 преобразование несколько усложняется. Как и следовало ожидать, функция возвращает `xs:boolean`, но при преобразовании аргумента используются следующие правила:

- Если аргумент является синглетом любого числового типа, `boolean()` возвращает `false`, если число является нулем или `NaN` («нечисло»); во всех остальных случаях возвращается `true`.
- Если аргумент является синглетом типа `xs:string`, `xs:anyURI`, `xs:untypedAtomic` или любого типа, производного от них, `boolean()` возвращает `true`, если аргумент имеет положительную длину.
- Если аргумент является синглетом типа `xs:boolean` (или типа, производного от `xs:boolean`), `boolean()` возвращает аргумент в неизменном виде.
- Если аргумент является последовательностью, первый объект которой представляет собой узел, `boolean()` возвращает `true`.
- Если аргумент является пустой последовательностью, `boolean()` возвращает `false`.
- Для любых других аргументов (например, `xs:date`, `xs:time` или последовательности из нескольких атомарных значений) `boolean()` выдает ошибку.

Где определяется

[1.0] XPath, раздел 4.3 «Boolean Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.1 «General Functions and Operators on Sequences».

Пример

Для демонстрации работы функции `boolean()` используется документ с информацией о продажах:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
```

```
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

А вот как выглядит таблица стилей:

```
<?xml version="1.0"?>
<!-- boolean.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;</xsl:text>
    <xsl:text>Tests of the boolean() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  boolean(true()) = </xsl:text>
    <xsl:value-of select="boolean(true())"/>

    <xsl:text>&#xA;  boolean(true) = </xsl:text>
    <xsl:value-of select="boolean(true)"/>

    <xsl:text>&#xA;  boolean('false') = </xsl:text>
    <xsl:value-of select="boolean('false')"/>

    <xsl:text>&#xA;  boolean('7') = </xsl:text>
    <xsl:value-of select="boolean('7')"/>

    <xsl:text>&#xA;  boolean(7) = </xsl:text>
    <xsl:value-of select="boolean(7)"/>

    <xsl:text>&#xA;  boolean(/report/brand/units[. > 20000]) = </xsl:text>
    <xsl:value-of select="boolean(/report/brand/units[. > 20000])"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the boolean() function:

  boolean(true()) = true
  boolean(true) = false
  boolean('false') = true
  boolean('7') = true
  boolean(7) = true
  boolean(/report/brand/units[. > 20000]) = true
```

В первом тесте в аргументе передается вызов функции `true()`**, который всегда возвращает** `true`**. Так как полученное значение уже является логическим, функция** `boolean()` **просто возвращает его в исходном виде. Во втором тесте запрашиваются все элемен-**

ты <true> в текущем контексте; так как элементы <true> отсутствуют, результат равен false. Аргументами третьего и четвертого тестов являются строки с положительной длиной, для которых boolean() возвращает true. В пятом тесте передается положительное число, поэтому результат также равен true.

В последнем примере команда XPath выбирает из отчета все элементы <units> с числовым значением, большим 20000. Так как команда выбирает по крайней мере один узел, boolean() возвращает true. Тест работает в XSLT 1.0, т. к. он генерирует набор узлов, содержащий хотя бы один узел, и в XSLT 2.0, т. к. он создает последовательность, первый объект которой является узлом. Если изменить выражение так, чтобы производился поиск объемов продаж, превышающих 30000, функция boolean() вернет false.

Следующая таблица стилей использует правила XSLT 2.0 для обработки различных последовательностей:

```
<?xml version="1.0"?>
<!-- boolean2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;</xsl:text>
    <xsl:text>Tests of the boolean() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  boolean(()) = </xsl:text>
    <xsl:value-of select="boolean(())"/>

    <xsl:variable name="testSequence1" as="item()*">
      <xsl:sequence select="(3)"/>
    </xsl:variable>

    <xsl:text>&#xA;&#xA;  $testSequence1 = (</xsl:text>
    <xsl:value-of select="$testSequence1" separator=", "/>
    <xsl:text>)&#xA;  boolean($testSequence1) = </xsl:text>
    <xsl:value-of select="boolean($testSequence1)"/>

    <xsl:variable name="testSequence2" as="item()*">
      <xsl:sequence select="/report/brand/units"/>
      <xsl:sequence select="(3, 4, 5)"/>
    </xsl:variable>

    <xsl:text>&#xA;&#xA;  $testSequence2 = (</xsl:text>
    <xsl:value-of select="$testSequence2" separator=", "/>
    <xsl:text>)&#xA;  boolean($testSequence2) = </xsl:text>
    <xsl:value-of select="boolean($testSequence2)"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the boolean() function:

  boolean(()) = false

  $testSequence1 = (3)
  boolean($testSequence1) = true

  $testSequence2 = (27408, 8203, 22101, 14336, 19268, 3, 4, 5)
  boolean($testSequence2) = true
```

Проверка состоит из трех тестов. В первом тесте функции `boolean()` передается пустая последовательность; результат равен `false`. Во втором тесте `boolean()` передается синглетное числовое значение; так как оно отлично от нуля, результат равен `true`. В последнем тесте аргумент представляет собой последовательность, содержащую пять узлов `<unit>`, за которыми следуют три числовых значения. Поскольку первый объект последовательности является узлом, `boolean()` возвращает `true`.

Если в этой таблице стилей определить `$testSequence1` так, чтобы последовательность содержала более одного объекта, таблица не будет компилироваться. Любая последовательность, передаваемая `boolean()`, должна быть синглетом или последовательностью, первое значение в которой является узлом. Во вторую последовательность `$testSequence2` можно включить сколько угодно объектов разных типов при условии, что первый объект является узлом. Если ни один узел не соответствует выражению `/report/brand/units`, первым узлом последовательности станет атомарное значение `3`, и процессор XSLT выдаст ошибку времени выполнения.

За дополнительными примерами и информацией обращайтесь к разделу «Преобразование к логическим значениям» главы 5.

ceiling()

Возвращает наименьшее целое, не меньшее аргумента.

Синтаксис

```
[1.0] number? ceiling(number?)
[2.0] numeric? ceiling(numeric?)
```

Ввод

Число.

[1.0] Если аргумент не является числом, он преобразуется в число так, как если бы он был обработан функцией `number()`. Если аргумент не удается преобразовать в число, функция `ceiling()` возвращает значение `NaN` («нечисло»).

[2.0] В XSLT 2.0 аргумент должен относиться к одному из четырех числовых типов (`xs:float`, `xs:decimal`, `xs:double` или `xs:integer`). В противном случае процессор XSLT выдает ошибку. Результат функции `ceiling()` относится к тому же типу, что и ее аргумент.

Вывод

Наименьшее целое, не меньшее аргумента.

[1.0] Если аргумент не может быть преобразован в число, в XSLT 1.0 функция `ceiling()` возвращает `NaN`.

[2.0] Если аргумент не может быть преобразован в число, в XSLT 2.0 функция `ceiling()` выдает ошибку.

Где определяется

[1.0] XPath, раздел 4.4 «Number Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 6.4 «Functions on Numeric Values».

Пример

Следующая таблица стилей демонстрирует результаты вызова функции `ceiling()` для разных значений. В качестве входных данных будет использоваться следующий документ XML:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Таблица стилей, использующая функцию `ceiling()`:

```
<?xml version="1.0"?>
<!-- ceiling.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the ceiling() function:&#xA;&#xA;</xsl:text>

    <xsl:text>  ceiling(7.983) = </xsl:text>
    <xsl:value-of select="ceiling(7.983)"/>

    <xsl:text>&#xA;  ceiling(-7.893) = </xsl:text>
    <xsl:value-of select="ceiling(-7.893)"/>

    <xsl:text>&#xA;  ceiling(avg(/report/brand/units)) = </xsl:text>
    <xsl:value-of select="ceiling(avg(/report/brand/units))"/>

    <xsl:text>&#xA;  ceiling('blue') = </xsl:text>
    <xsl:value-of version="1.0" select="ceiling('blue')"/>

  </xsl:template>

</xsl:stylesheet>
```

Результат преобразования документа XML по таблице стилей:

```
Tests of the ceiling() function:

  ceiling(7.983) = 8
  ceiling(-7.893) = -7
  ceiling(avg(/report/brand/units)) = 18264
  ceiling('blue') = NaN
```

В последнем тесте в последнем элементе <xsl:value-of> мы указали версию "1.0". В режиме XSLT 1.0 функция возвращает результат NaN («нечисло»). При попытке обработать этот элемент <xsl:value-of> в режиме XSLT 2.0 происходит ошибка.

[2.0] codepoint-equal()

Определяет, равны ли две строки, посредством сопоставления кодов символов в Юникоде.

Синтаксис

```
xs:boolean? codepoint-equal(xs:string?, xs:string?)
```

Ввод

Две сравниваемые строки xs:string.

Вывод

Функция возвращает true или false в зависимости от того, равны ли две строки при сопоставлении кодов символов в Юникоде. Функция позволяет сравнивать любые значения xs:anyURI без определения схем упорядочения. Если хотя бы один аргумент является пустой последовательностью, результатом будет также пустая последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.3 «Equality and Comparison of Strings».

Пример

Короткая таблица стилей для проверки равенства строк:

```
<?xml version="1.0"?>
<!-- codepoint-equal.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the codepoint-equal() function:&#xA;</xsl:text>

    <xsl:text>&#xA;  codepoint-equal('A', 'A') = </xsl:text>
    <xsl:value-of select="codepoint-equal('A', 'A')"/>

    <xsl:text>&#xA;  codepoint-equal('A', '&amp;#x41;') = </xsl:text>
    <xsl:value-of select="codepoint-equal('A', '&#x41;')"/>

    <xsl:text>&#xA;  codepoint-equal('A', '&amp;#65;') = </xsl:text>
```

```
    <xsl:value-of select="codepoint-equal('A', '&#65;')"/>

    <xsl:text>&#xA;  codepoint-equal('Strasse', 'Stra&amp;#xDF;e') = </xsl:text>
    <xsl:value-of select="codepoint-equal('Strasse', 'Stra&#xDF;e')"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the codepoint-equal() function:

  codepoint-equal('A', 'A') = true
  codepoint-equal('A', '&#x41;') = true
  codepoint-equal('A', '&#65;') = true
  codepoint-equal('Strasse', 'Straße') = false
```

В первых трех тестах прописная буква A сравнивается сама с собой и со своими шестнадцатеричным и десятичным представлениями. В последнем тесте сравниваются два варианта написания немецкого слова *Strasse*: с символами *ss* и с немецким «длинным s» (ß). При сопоставлении кодов Юникода эти две строки не равны.

В XPath 2.0 и XSLT 2.0 определяются функции, позволяющие применять разные схемы упорядочения (скажем, схему, в которой ss и ß считаются равными). За примерами обращайтесь к описаниям функций *[2.0]* `compare()`, `contains()`, *[2.0]* `deep-equal()`, *[2.0]* `distinct-values()`, *[2.0]* `ends-with()`, *[2.0]* `index-of()`, *[2.0]* `max()`, *[2.0]* `min()`, `starts-with()`, `substring-after()` и `substring-before()`.

[2.0] codepoints-to-string()

Преобразует последовательность кодов символов в Юникоде в строку.

Синтаксис

```
xs:string codepoints-to-string(xs:integer*)
```

Ввод

Последовательность значений `xs:integer`, каждое из которых представляет код символа в Юникоде.

Вывод

Значение `xs:string` создается посредством конкатенации заданных кодов символов. Если аргумент является пустой последовательностью, `codepoints-to-string()` возвращает строку нулевой длины. Если какой-либо из кодов символов в аргументе не является допустимым символом XML, процессор XSLT выдает ошибку.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.2 «Functions to Assemble and Disassemble Strings».

Пример

Короткая таблица стилей для преобразования последовательности целых чисел в строки:

```
<?xml version="1.0"?>
<!-- codepoints-to-string.xsl -->
```

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the codepoints-to-string() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  codepoints-to-string</xsl:text>
    <xsl:text>((76, 105, 108, 121)) = </xsl:text>
    <xsl:value-of
      select="codepoints-to-string((76, 105, 108, 121))"/>

    <xsl:text>&#xA;  codepoints-to-string</xsl:text>
    <xsl:text>((83, 116, 114, 97, 223, 101)) = </xsl:text>
    <xsl:value-of
      select="codepoints-to-string((83, 116, 114, 97, 223, 101))"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the codepoints-to-string() function:

  codepoints-to-string((76, 105, 108, 121)) = Lily
  codepoints-to-string((83, 116, 114, 97, 223, 101)) = Straße
```

Как упоминалось ранее, если хотя бы один код символа во входной последовательности не является действительным символом XML, процессор XSLT выдает ошибку. Например, вызов `codepoints-to-string((0))` приводит к сбою работы процессора XSLT; вы можете самостоятельно убедиться в этом на досуге.

[2.0] collection()

Возвращает коллекцию узлов; расширяет возможности функции `document()`, определенной в XSLT 1.0.

Синтаксис

```
node()* collection(xs:string?)
```

Ввод

Необязательное значение `xs:string` с URI-адресом коллекции. Если строка не указана, возвращается коллекция по умолчанию.

Вывод

Заданная коллекция узлов. Если имя коллекции не задано, возвращается коллекция по умолчанию. Подробности формата URI и типы ресурсов, к которым возможны определения, зависят от реализации.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.5 «Functions and Operators that Generate Sequences».

Пример

В нашем документе будет использоваться реализация Saxon, которая позволяет загрузить документ XML, содержащий ссылки на другие документы XML. Документ коллекции имеет следующую структуру:

```
<?xml version="1.0"?>
<!-- polist.xml -->
<collection>
  <doc href="po38292.xml"/>
  <doc href="po38293.xml"/>
  <doc href="po38294.xml"/>
  <doc href="po38295.xml"/>
</collection>
```

При загрузке документа функцией `collection()` Saxon просматривает атрибуты `href` всех элементов `<doc>` и интерпретирует каждый из них как URI-адрес документа. Затем Saxon открывает и разбирает эти документы, добавляя их узлы в коллекцию. Каждый URI-адрес документа в нашем примере ссылается на документ с заказом на покупку, который имеет следующую структуру:

```
<?xml version="1.0" ?>
<!-- po38293.xml -->
<purchase-order id="38293">
  <date year="2001" month="9" day="8"/>
  <customer id="4738" level="Basic">
    <address type="business">
      <name>
        <title>Ms.</title>
        <first-name>Amanda</first-name>
        <last-name>Reckonwith</last-name>
      </name>
      <street>930-A Chestnut Street</street>
      <city>Lynn</city>
      <state>MA</state>
      <zip>02930</zip>
    </address>
    <address type="ship-to"/>
  </customer>
  <items>
    <item part-no="23813-03-CDK">
      <name>Cucumber Decorating Kit</name>
      <qty>1</qty>
      <price>29.95</price>
    </item>
  </items>
</purchase-order>
```

При обработке файла *polist.xml* коллекция содержит четыре элемента документа, каждый из которых представляет отдельный заказ на покупку. Далее XSLT и XPath используются для работы со всеми узлами коллекции. Следующая таблица стилей загружает коллекцию и работает с ней:

```
<?xml version="1.0"?>
<!-- collection.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the collection() function:</xsl:text>

    <xsl:variable name="docPile" as="node()*"
      select="collection('polist.xml')"/>

    <xsl:text>&#xA;&#xA;  The customers in the </xsl:text>
    <xsl:text>collection are: &#xA;    </xsl:text>
    <xsl:for-each select="$docPile/purchase-order/customer">
      <xsl:sort select="address/name/last-name"/>
      <xsl:value-of
        select="address/name/title,
                address/name/first-name,
                address/name/last-name"
        separator=" "/>
      <xsl:text>&#xA;    </xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

При обработке таблицы стилей будет получен следующий результат:

```
A test of the collection() function:

  The customers in the collection are:
    Ms. Natalie Attired
    Mrs. Mary Backstayge
    Mr. Chester Hasbrouck Frisby
    Ms. Amanda Reckonwith
```

Мы загружаем коллекцию и сохраняем узлы документов всех файлов в переменной `docPile`. После загрузки коллекции мы можем работать с ее узлами, несмотря на то, что они не имеют общего корня документа. В таблице стилей из нашего примера перечисляются все покупатели из всех заказов на покупку в коллекции. Элемент `<xsl:for-each>` используется для перебора всех заказов, а элемент `<xsl:sort>` упорядочивает покупателей по фамилиям.

Помните, что большинство деталей в работе функции `collection()` зависит от реализации. Использование элементов `<collection>` и `<doc>` в Saxon не оговорено в спецификации, как и возможность загрузки документа коллекции по имени файла. За информацией о реализации функции `collection()` обращайтесь к документации своего процессора XSLT.

[2.0] compare()

Сравнивает два значения `xs:string` и возвращает `-1`, `0` или `1`, если первая строка соответственно меньше, равна или больше второй строки.

Синтаксис

```
xs:integer? compare(xs:string?, xs:string?)
xs:integer? compare(xs:string?, xs:string?, $collation as xs:string)
```

Ввод

Два значения `xs:string` и необязательная схема упорядочения.

Вывод

Числовое значение -1, 0 или 1 (первая строка соответственно меньше, равна или больше второй строки). Необязательный аргумент $collation задает имя схемы упорядочения, используемой для сравнения двух строк. Воспользуемся примером из спецификации XSLT 2.0: в схеме упорядочения для немецкого языка строки *Strasse* и *Straße* считаются равными, в других языках они различаются. Если хотя бы одна из строк является пустой последовательностью, compare() возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.3 «Equality and Comparison of Strings».

Пример

Следующая таблица стилей демонстрирует разные варианты использования compare():

```
<?xml version="1.0"?>
<!-- compare1.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Here are some tests of the compare() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  compare('Lily', 'lily') = </xsl:text>
    <xsl:value-of select="compare('Lily', 'lily')"/>

    <xsl:text>&#xA;  compare('Lily', 'Lily') = </xsl:text>
    <xsl:value-of select="compare('Lily', 'Lily')"/>

    <xsl:text>&#xA;  compare('Lily', </xsl:text>
    <xsl:text>'&amp;#x4C;&amp;#x69;&amp;#x6C;&amp;#x79;') = </xsl:text>
    <xsl:value-of select="compare('Lily', '&#x4C;&#x69;&#x6C;&#x79;')"/>

    <xsl:text>&#xA;  compare('Lily', 'Doug') = </xsl:text>
    <xsl:value-of select="compare('Lily', 'Doug')"/>
    <xsl:text>&#xA;&#xA;  if (not(compare('Lily', 'Lily'))) : </xsl:text>
    <xsl:value-of select="if (not(compare('Lily', 'Lily')))
                          then 'The two test strings are equal!'
                          else 'The two test strings aren''t equal!'"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here are some tests of the compare() function:

  compare('Lily', 'lily') = -1
  compare('Lily', 'Lily') = 0
  compare('Lily', '&#x4C;&#x69;&#x6C;&#x79;') = 0
  compare('Lily', 'Doug') = 1

  if (not(compare('Lily', 'Lily'))) : The two test strings are equal!
```

В первых четырех примерах функция compare() сравнивает строки, причем в третьем примере текст заменяется символьными сущностями. В пятом примере функция compare() используется в логическом выражении: если две строки равны, compare() возвращает 0. Выражение not(compare()) всего лишь является неуклюжим представлением условия «Если эти две строки не являются неравными».

Помните, что в XPath могут использоваться любые операторы сравнения – как операторы для атомарных значений (eq, ne, gt, lt, ge и le), так и более общие операторы (=, !=, >, <, >= и <=). Функция compare() действительно необходима только при задании схемы упорядочения.

Раз уж мы затронули эту тему, рассмотрим последний пример, в котором используется схема упорядочения. В немецком языке слово «улица» может записываться в двух вариантах: *Strasse* и *Straße*. Мы дважды сравним эти два слова – с использованием стандартной схемы упорядочения и пользовательской функции для немецкого языка. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- compare2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="string1" select="'Stra&#xDF;e'"/>
    <xsl:variable name="string2" select="'Strasse'"/>

    <xsl:text>&#xA;Here are more tests of the compare() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>  compare('</xsl:text>
    <xsl:value-of select="$string1"/>
    <xsl:text>', '</xsl:text>
    <xsl:value-of select="$string2"/>
    <xsl:text>') = </xsl:text>
    <xsl:value-of select="compare($string1, $string2)"/>
    <xsl:text>&#xA;</xsl:text>
    <xsl:text>  compare('</xsl:text>
    <xsl:value-of select="$string1"/>
    <xsl:text>', '</xsl:text>
    <xsl:value-of select="$string2"/>
    <xsl:text>', [German collation]) = </xsl:text>
    <xsl:value-of
      select="compare($string1, $string2,
              concat('http://saxon.sf.net/collation?',
                     'class=com.oreilly.xslt.GermanCollation;'))"/>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here are more tests of the compare() function:
  compare('Straße', 'Strasse') = 1
  compare('Straße', 'Strasse', [German collation]) = 0
```

Как видите, два варианта написания различаются при стандартном упорядочении, но считаются равными при использовании упорядочения для немецкого языка. (Чтобы листинг поместился на странице, мы объединили две половины URI-адреса схемы упорядочения Saxon при помощи функции `concat()`.) За дополнительной информацией обращайтесь к разделу «Функция document() и сортировка» главы 8.

concat()

Получает все аргументы и объединяет их посредством конкатенации. Аргументы, не являющиеся строками, преобразуются в строки, как если бы они были обработаны функцией `string()`.

Синтаксис

```
[1.0] string concat(string, string, string*)
[2.0] xs:string concat(xs:anyAtomicType?, xs:anyAtomicType?, ...)
```

Ввод

Две или более строки. Функция `concat()` отличается от других функций тем, что она может иметь произвольное количество аргументов.

[2.0] В XSLT 2.0 аргументы, передаваемые функции `concat()`, могут быть произвольными атомарными значениями. Все значения преобразуются в строки. Если хотя бы один аргумент является пустой последовательностью, он интерпретируется как строка нулевой длины. Если хотя бы один аргумент является последовательностью, длина которой превышает 1, происходит ошибка.

Вывод

Результат конкатенации всех входных значений.

Где определяется

[1.0] XPath, раздел 4.2 «String Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.4 «Functions on String Values».

Пример

Для демонстрации работы `concat()` используется следующий файл XML:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

В таблице стилей функция `concat()` **используется для создания имен различных файлов HTML. Имена файлов строятся из трех компонентов, объединяемых функцией** `concat()`**:**

```
<?xml version="1.0"?>
<!-- concat.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the concat() function: &#xA;</xsl:text>
    <xsl:for-each select="list/listitem">
      <xsl:text>&#xA;  See the file </xsl:text>
      <xsl:value-of select="concat('album', position(), '.html')"/>
      <xsl:text> to see the details of &#xA;    the album "</xsl:text>
      <xsl:value-of select="."/>
      <xsl:text>."&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Tests of the concat() function:

  See the file album1.html to see the details of
    the album "The Sacred Art of Dub."

  See the file album2.html to see the details of
    the album "Only the Poor Man Feel It."

  See the file album3.html to see the details of
    the album "Excitable Boy."
  ...
```

[2.0] **Следующая таблица стилей демонстрирует объединение атомарных значений функцией** `concat()` **в XSLT 2.0:**

```
<?xml version="1.0"?>
<!-- concat2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Another test of the concat() function: &#xA;</xsl:text>

    <xsl:text>&#xA;  concat(3, 4, current-time(), 'blue', </xsl:text>
    <xsl:text>/list/title) = &#xA;    </xsl:text>
    <xsl:value-of
      select="concat(3, 4, current-time(), 'blue', /list/title)"/>
  </xsl:template>

</xsl:stylesheet>
```

Для нашего входного документа XML будет получен следующий результат:

```
Another test of the concat() function:

  concat(3, 4, current-time(), 'blue', /list/title) =
    3416:23:12.217-04:00blueAlbums I've bought recently:
```

Функция объединяет числа 3 и 4, возвращаемое функцией current-time() значение xs:time, строку blue и элемент <title> из входного документа. Все эти значения преобразуются в строки и объединяются посредством конкатенации.

Все аргументы concat() должны быть атомарными значениями или последовательностями, содержащими нуль или одно значение. Таблица стилей работает со входным документом XML, потому что документ содержит только один узел, соответствующий выражению XPath /list/title. Если изменить выражение XPath на /list/listitem, оно выберет более одного узла, и процессор XSLT выдаст ошибку. Для объединения последовательностей из нескольких объектов следует использовать функцию *[2.0]* string-join().

contains()

Определяет, содержится ли второй строковый аргумент в первом.

Синтаксис

```
[1.0] boolean contains(string, string)
[2.0] xs:boolean contains(xs:string?, xs:string?)
[2.0] xs:boolean contains(xs:string?, xs:string?, $collation as xs:string)
```

Ввод

Две строки. Если первая строка содержит вторую, функция возвращает логическое значение true. *[2.0]* В XSLT 2.0 функции может передаваться необязательный третий аргумент – имя схемы упорядочения, определяющей способ сравнения строк для конкретного языка.

Вывод

Логическое значение true, если первый аргумент содержит второй; false в противном случае. Если вторая строка имеет нулевую длину, contains() возвращает true. Если первая строка имеет нулевую длину, contains() возвращает false.

Где определяется

[1.0] XPath, раздел 4.2 «String Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.5 «Functions Based on Substring Matching».

Пример

Таблица стилей использует шаблон с именем replace-substring. Шаблону replace-substring передаются три аргумента: исходная строка; подстрока, которая ищется в исходной строке; и подстрока, заменяющая искомую подстроку в исходной сроке. Шаблон replace-substring использует функции contains(), substring-after() и substring-before().

Следующая таблица стилей заменяет все вхождения подстроки `World` подстрокой `"Mundo"`:

```
<?xml version="1.0"?>
<!-- contains1.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the contains() function:&#xA;&#xA;</xsl:text>
    <xsl:text>  Replacing 'World' with 'Mundo' in </xsl:text>
    <xsl:text>'Hello World!': &#xA;    </xsl:text>
    <xsl:variable name="test">
      <xsl:call-template name="replace-substring">
        <xsl:with-param name="original">Hello World!</xsl:with-param>
        <xsl:with-param name="substring">World</xsl:with-param>
        <xsl:with-param name="replacement">Mundo</xsl:with-param>
      </xsl:call-template>
    </xsl:variable>
    <xsl:value-of select="$test"/>
  </xsl:template>

  <xsl:template name="replace-substring">
    <xsl:param name="original" />
    <xsl:param name="substring" />
    <xsl:param name="replacement" />
    <xsl:choose>
      <xsl:when test="contains($original, $substring)">
        <xsl:value-of select="substring-before($original, $substring)" />
        <xsl:value-of select="$replacement" />
        <xsl:call-template name="replace-substring">
          <xsl:with-param name="original"
            select="substring-after($original, $substring)" />
          <xsl:with-param name="substring" select="$substring" />
          <xsl:with-param name="replacement" select="$replacement" />
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$original" />
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

</xsl:stylesheet>
```

В этом примере одна подстрока заменяется другой. Для решения этой задачи мы используем три функции (`contains()`, `substring-after()` и `substring-before()`) и рекурсивный именованный шаблон (`replace-substring`). Рекурсивный шаблон получает три строки: исходную строку, заменяемую строку и заменяющую строку. Если исходная строка не содержит ни одного вхождения заменяемой строки, шаблон возвращает всю строку. В противном случае шаблон рекурсивно вызывает сам себя с тремя аргументами: частью строки после заменяемой строки, заменяемой и заменяющей строками. Другими словами, если именованный шаблон находит заменяемую строку, он вызывает себя для остатка строки. Функция `substring-after` играет ключевую роль в работе шаблона.

Таблица стилей выдает следующий результат (независимо от того, какой документ XML передается ей на входе):

```
Hello Mundo!
```

[2.0] В первом примере используется XSLT 1.0, поэтому для замены подстрок нам приходится использовать рекурсивную именованную процедуру, использующую функции `contains()`, `substring-after()` и `substring-before()`. В XSLT 2.0 вместо этого можно воспользоваться очень удобной функцией `replace()`:

```
<?xml version="1.0"?>
<!-- replace2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the replace() function:&#xA;&#xA;</xsl:text>
    <xsl:text>  Replacing 'World' with 'Mundo' in </xsl:text>
    <xsl:text>'Hello World!': &#xA;    </xsl:text>
    <xsl:value-of select="replace('Hello World', 'World', 'Mundo')"/>
  </xsl:template>

</xsl:stylesheet>
```

Эта таблица стилей выдает тот же результат, что и таблица стилей XSLT 1.0, но имеет гораздо меньший объем кода.

Последний пример демонстрирует использование схемы упорядочения:

```
<?xml version="1.0"?>
<!-- contains2.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="string1"
      select="'Sch&#xF6;naicherstra&#xDF;e'"/>
    <xsl:variable name="string2" select="'strasse'"/>

    <xsl:text>&#xA;Another test of the contains() </xsl:text>
    <xsl:text>function:&#xA;&#xA;</xsl:text>

    <xsl:text>  contains('</xsl:text>
    <xsl:value-of select="$string1"/>
    <xsl:text>', '</xsl:text>
    <xsl:value-of select="$string2"/>
    <xsl:text>') = </xsl:text>
    <xsl:value-of select="contains($string1, $string2)"/>
    <xsl:text>&#xA;</xsl:text>

    <xsl:text>  contains('</xsl:text>
    <xsl:value-of select="$string1"/>
    <xsl:text>', '</xsl:text>
    <xsl:value-of select="$string2"/>
    <xsl:text>', [German collation]) = </xsl:text>
    <xsl:value-of
      select="contains($string1, $string2,
```

```
            concat('http://saxon.sf.net/collation?',
                   'class=com.oreilly.xslt.GermanCollation;'))"/>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Another test of the contains() function:

  contains('Schönaicherstraße', 'strasse') = false
  contains('Schönaicherstraße', 'strasse', [German collation]) = true
```

В немецком языке слово «улица» может записываться в двух вариантах: *Strasse* и *Straße*. Немецкая схема упорядочения в этой таблице стилей учитывает существование этих двух вариантов, упорядочение по умолчанию – нет. (Чтобы листинг поместился на странице, мы объединили две половины URI-адреса схемы упорядочения Saxon при помощи функции `concat()`.) За дополнительной информацией обращайтесь к разделу «Функция document() и сортировка» главы 8.

count()

Подсчитывает количество узлов в *[1.0]* наборе узлов или *[2.0]* последовательности.

Синтаксис

```
[1.0] number count(node-set)
[2.0] xs:integer count(item()*)
```

Ввод

Набор узлов или последовательность.

Вывод

Количество объектов в наборе узлов или последовательности. *[2.0]* Если аргумент является пустой последовательностью, `count()` возвращает `0`.

Где определяется

[1.0] XPath, раздел 4.1 «Node Set Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.4 «Aggregate Functions».

Пример

Для демонстрации работы функции `count()` используется следующий документ XML:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
```

```
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Таблица стилей для демонстрации работы функции `count()`**:**

```
<?xml version="1.0"?>
<!-- count.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the count() function:&#xA;&#xA;</xsl:text>
    <xsl:text>  Our store sells </xsl:text>
    <xsl:value-of select="count(/report/brand)"/>
    <xsl:text> different brands of chocolate.</xsl:text>
    <xsl:text>&#xA;&#xA;  For the last month, </xsl:text>
    <xsl:value-of select="count(/report/brand[units > 20000])"/>
    <xsl:text> brand(s) sold more than 20,000 units,</xsl:text>
    <xsl:text>&#xA;    and all but </xsl:text>
    <xsl:value-of select="count(/report/brand[units &lt; 10000])"/>
    <xsl:text> brand(s) sold 10,000 units or more.</xsl:text>

    <xsl:text>&#xA;&#xA;  Here are the brands we sell: </xsl:text>
    <xsl:for-each select="/report/brand">
      <xsl:text>&#xA;    Brand </xsl:text>
      <xsl:value-of select="position()"/>
      <xsl:text> of </xsl:text>
      <xsl:value-of select="count(/report/brand)"/>
      <xsl:text>: </xsl:text>
      <xsl:value-of select="name"/>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the count() function:

  Our store sells 5 different brands of chocolate.

  For the last month, 2 brand(s) sold more than 20,000 units,
    and all but 1 brand(s) sold 10,000 units or more.

  Here are the brands we sell:
```

```
Brand 1 of 5: Lindt
Brand 2 of 5: Callebaut
Brand 3 of 5: Valrhona
Brand 4 of 5: Perugina
Brand 5 of 5: Ghirardelli
```

В таблице представлено несколько вариантов использования функции `count()`. В первой фразе число `5` является количеством элементов `<brand>` в документе. Во второй фразе функция `count()` используется с предикатом XPath для подсчета только тех элементов `<brand>`, у которых дочерние элементы `<units>` содержат определенные значения. Наконец, в списке продаваемых брендов функции `position()` и `count()` совместно используются для построения текста «Brand x of y».

Пара обстоятельств заслуживает особого внимания. Во-первых, таблица стилей начинается с элемента `<xsl:stylesheet version="1.0">`, поэтому она работает в процессорах XSLT 1.0 и 2.0. Во-вторых, знак «меньше» (`<`) заменяется сущностью `<`, так как этот знак не может содержаться в значении атрибута.

Следующая короткая таблица стилей демонстрирует работу функции `count()` с последовательностями XSLT 2.0:

```
<?xml version="1.0"?>
<!-- count2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the count() function:&#xA;&#xA;</xsl:text>

    <xsl:variable name="salesFigures" as="xs:integer*">
      <xsl:sequence select="/report/brand/units"/>
    </xsl:variable>

    <xsl:text>  Our store sells </xsl:text>
    <xsl:value-of select="count($salesFigures)"/>
    <xsl:text> different brands of chocolate.</xsl:text>
    <xsl:text>&#xA;&#xA;  For the last month, </xsl:text>
    <xsl:value-of select="count($salesFigures[. > 20000])"/>
    <xsl:text> brand(s) sold more than 20,000 units,</xsl:text>
    <xsl:text>&#xA;    and all but </xsl:text>
    <xsl:value-of select="count($salesFigures[. &lt; 10000])"/>
    <xsl:text> brand(s) sold 10,000 units or more.</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Мы создаем последовательность `xs:integer` и используем функцию `count()` для подсчета объектов в последовательности. Как и в первой таблице стилей, для отбора значений из последовательности по некоторому критерию применяются предикаты. Результат выглядит так:

```
Tests of the count() function:

  Our store sells 5 different brands of chocolate.

  For the last month, 2 brand(s) sold more than 20,000 units,
    and all but 1 brand(s) sold 10,000 units or more.
```

Во второй таблице стилей последовательность содержит значения типа `xs:integer` из элементов `<units>` исходного документа, поэтому предикат XPath использует текущий узел **(.)** в выражении сравнения: `count($salesFigures[. > 20000]`. В первой таблице стилей выражение `count(/report/brand[units < 10000])` можно было записать и в форме `count(/report/brand/units[. < 10000]`; результат от этого не изменился бы.

current()

Возвращает набор узлов, состоящий только из текущего узла.

Синтаксис

```
[1.0] node-set current()
[2.0] item() current()
```

Ввод

Нет.

Вывод

[1.0] Набор узлов, состоящий только из текущего узла. В большинстве случаев текущий узел совпадает с контекстным узлом.

[2.0] В выражении XPath эта функция возвращает объект, который был контекстным узлом на момент активизации выражения.

В XSLT 1.0 и 2.0 текущий узел/объект почти всегда совпадает с контекстным узлом/объектом. Например, следующие два выражения XPath почти всегда совпадают:

```
<xsl:value-of select="current()"/>
<xsl:value-of select="."/>
```

Однако в предикатных выражениях текущий узел обычно отличается от контекстного.

Где определяется

[1.0] XSLT, раздел 12.4 «Miscellaneous Additional Functions».

[2.0] XSLT, раздел 16.6 «Miscellaneous Additional Functions».

Пример

В следующем примере функция `current()` используется с таблицей преобразования. В качестве источника данных используется видоизмененная версия документа с данными о продажах шоколада:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate-by-month.xml -->
<report year="2006">
  <title>Chocolate bar sales by month</title>
  <sales month="08" total="91316"/>
  <sales month="09" total="82911"/>
  <sales month="10" total="128587"/>
  <sales month="11" total="79244"/>
  <sales month="12" total="113606"/>
</report>
```

А вот как выглядит таблица стилей. Она представляет собой таблицу преобразования, которая ставит в соответствие атрибуту `month` название надлежащего месяца. Таблица стилей выполняет одно преобразование трижды – один раз с функцией `current()` и два раза без нее:

```
<?xml version="1.0"?>
<!-- current.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:months="http://www.months.com">

  <months:name sequence="01">January</months:name>
  <months:name sequence="02">February</months:name>
  <months:name sequence="03">March</months:name>
  <months:name sequence="04">April</months:name>
  <months:name sequence="05">May</months:name>
  <months:name sequence="06">June</months:name>
  <months:name sequence="07">July</months:name>
  <months:name sequence="08">August</months:name>
  <months:name sequence="09">September</months:name>
  <months:name sequence="10">October</months:name>
  <months:name sequence="11">November</months:name>
  <months:name sequence="12">December</months:name>

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the current() function:&#xA;</xsl:text>

    <xsl:text>&#xA;  Using the current() function:&#xA;</xsl:text>
    <xsl:text>&#xA;    Chocolate bar sales for </xsl:text>
    <xsl:value-of select="/report/@year"/>
    <xsl:text>: &#xA;</xsl:text>

    <xsl:for-each select="/report/sales">
      <xsl:text>&#xA;      </xsl:text>
      <xsl:value-of
        select="document('')/*/months:name[@sequence=current()/@month][1]"/>
      <xsl:text> - Total sales = </xsl:text>
      <xsl:value-of select="format-number(@total, '##,###')"/>
    </xsl:for-each>

    <xsl:text>&#xA;&#xA;  Without the current() function:&#xA;</xsl:text>
    <xsl:text>&#xA;    Chocolate bar sales for </xsl:text>
    <xsl:value-of select="/report/@year"/>
    <xsl:text>: &#xA;</xsl:text>

    <xsl:for-each select="/report/sales">
      <xsl:text>&#xA;      </xsl:text>
      <xsl:value-of
        select="document('')/*/months:name[@sequence=./@month][1]"/>
      <xsl:text> - Total sales = </xsl:text>
      <xsl:value-of select="format-number(@total, '##,###')"/>
    </xsl:for-each>

    <xsl:text>&#xA;&#xA;  Another test without the current() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>
    <xsl:text>&#xA;    Chocolate bar sales for </xsl:text>
    <xsl:value-of select="/report/@year"/>
    <xsl:text>: &#xA;</xsl:text>
```

```
    <xsl:for-each select="/report/sales">
      <xsl:text>&#xA;      </xsl:text>
      <xsl:value-of
        select="document('')/*/months:name[@sequence=./@sequence][1]"/>
      <xsl:text> - Total sales = </xsl:text>
      <xsl:value-of select="format-number(@total, '##,###')"/>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the current() function:

  Using the current() function:

    Chocolate bar sales for 2006:

      August - Total sales = 91,316
      September - Total sales = 82,911
      October - Total sales = 128,587
      November - Total sales = 79,244
      December - Total sales = 113,606

  Without the current() function:

    Chocolate bar sales for 2006:

       - Total sales = 91,316
       - Total sales = 82,911
       - Total sales = 128,587
       - Total sales = 79,244
       - Total sales = 113,606

  Another test without the current() function:

    Chocolate bar sales for 2006:

      January - Total sales = 91,316
      January - Total sales = 82,911
      January - Total sales = 128,587
      January - Total sales = 79,244
      January - Total sales = 113,606
```

Первый тест работает правильно благодаря использованию функции `current()`. Функция возвращает элемент `<sales>`, обрабатываемый в данный момент. В предикатном выражении для получения этого элемента приходится использовать функцию `current()`. Элемент `<xsl:value-of>` выводит текст элемента `<months:name>`, у которого атрибут `sequence` соответствует атрибуту `month` текущего элемента `<sales>`.

Обратите внимание: выражение XPath завершается еще одним предикатом `[1]`, который обеспечивает выбор только первого совпадения. В XSLT 1.0 это несущественно, поскольку `<xsl:value-of>` использует только первый узел из набора узлов, но в XSLT 2.0 используются *все* совпавшие узлы. Из-за структуры нашего исходного документа XML в первом примере совпадает только один объект, однако для произвольного исходного документа XML это условие может и не выполняться. По этой причине желательно точно указывать, какой узел вы хотите использовать, особенно если таблица стилей XSLT 1.0 позднее будет переноситься в XSLT 2.0.

Второй и третий тесты работают неправильно, так как в них не используется функция current(). Во втором тесте выражение XPath имеет вид document('')/*/months:name[@sequence=./@month][1]. Точка представляет обрабатываемый элемент <months:name>, а не текущий элемент <sales>. Предикат XPath [@sequence=./@month] не совпадает ни с чем, так как в документе отсутствуют элементы <months:name> с атрибутом month.

Задача третьего теста – показать, что же в действительности представляет оператор «точка» (контекстный узел). Если бы он представлял элемент <sales>, обрабатываемый в данный момент, предикат [@sequence=./@sequence] не совпал бы ни с чем; в документе нет элементов <sales> с атрибутом sequence. Но поскольку точка представляет элемент <months:name>, совпадение находится. В XSLT указано, что выводиться должен первый объект ([1]), поэтому в каждой строке выводится месяц January.

И последнее замечание по поводу предиката, указывающего на использование первого объекта: в первом тесте предикат [1] можно было опустить, так как в данном исходном документе XML выражение XPath совпадает только с одним узлом. Во втором тесте предикат [1] тоже не нужен, так как совпадения отсутствуют. Однако в третьем тесте попытка опустить предикат и указать в таблице стилей версию 2.0 приводит к следующему результату:

```
    Another test without the current() function:

      Chocolate bar sales for 2006:

        January February March April May June July August September Octo
ber November December - Total sales = 91,316
        January February March April May June July August September Octo
ber November December - Total sales = 82,911
        January February March April May June July August September Octo
ber November December - Total sales = 128,587
        January February March April May June July August September Octo
ber November December - Total sales = 79,244
        January February March April May June July August September Octo
ber November December - Total sales = 113,606
```

В XSLT 2.0 элемент <xsl:value-of> использует все узлы, соответствующие шаблону, а не только первый из них.

[2.0] current-date()

Возвращает текущую дату в виде значения xs:date.

Синтаксис

```
xs:date current-date()
```

Ввод

Нет.

Вывод

Значение типа xs:date, заполненное текущей датой системы. В возвращаемом значении устанавливается часовой пояс по умолчанию. (Для получения информации о часовом поясе по умолчанию используется функция implicit-timezone().) *Функция* current-date() *возвращает одинаковые значения в процессе обработки таблицы стилей*. Даже если функция current-date() десятикратно вызывается в таблице стилей, каждый раз она возвращает одно и то же значение.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 16 «Context Functions».

Пример

Следующая таблица стилей отображает текущую дату. В ней используются функции `current-date()`, `format-date()` и `implicit-timezone()`.

```
<?xml version="1.0"?>
<!-- current-date.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>The current date is </xsl:text>
    <xsl:value-of
      select="format-date(current-date(),
              '[FNn], the [D1o] of [MNn], [YO1]')"/>
    <xsl:text>&#xA;&#xA;The implicit timezone for the </xsl:text>
    <xsl:text>current context is: </xsl:text>
    <xsl:value-of select="implicit-timezone()"/>
    <xsl:text>&#xA;&#xA;The timezone extracted from the </xsl:text>
    <xsl:text>current date is: </xsl:text>
    <xsl:value-of select="timezone-from-date(current-date())"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
The current date is Thursday, the 6th of July, 06

The implicit timezone for the current context is: -PT4H

The timezone extracted from the current date is: -PT4H
```

Обратите внимание: часовой пояс по умолчанию, полученный функцией `implicit-timezone()`, совпадает с часовым поясом, извлеченным из текущей даты. *Часовой пояс по умолчанию* является компонентом всех значений `xs:date`, `xs:dateTime` и `xs:time`, создаваемых без указания конкретного часового пояса.

[2.0] current-dateTime()

Возвращает текущую дату и время в виде значения `xs:dateTime`.

Синтаксис

```
xs:dateTime current-dateTime()
```

Ввод

Нет.

Вывод

Значение типа `xs:dateTime`, заполненное текущей датой и временем системы. В возвращаемом значении устанавливается часовой пояс по умолчанию. (Для получения

информации о часовом поясе по умолчанию используется функция implicit-timezone().) *Функция* current-dateTime() *возвращает одинаковые значения в процессе обработки таблицы стилей.* Даже если функция current-dateTime() десятикратно вызывается в таблице стилей, каждый раз она возвращает одно и то же значение.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 16 «Context Functions».

Пример

Следующая таблица стилей отображает текущую дату и время. В ней используются функции current-dateTime(), format-dateTime() и implicit-timezone().

```
<?xml version="1.0"?>
<!-- current-datetime.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>The current date and time is </xsl:text>
    <xsl:value-of
      select="format-dateTime(current-dateTime(),
              '[h]:[m01] [Pn] on [FNn], the [D1o] of [MNn], [Y]')"/>
    <xsl:text>&#xA;&#xA;The implicit timezone for the </xsl:text>
    <xsl:text>current context is: </xsl:text>
    <xsl:value-of select="implicit-timezone()"/>
    <xsl:text>&#xA;&#xA;The timezone extracted from the </xsl:text>
    <xsl:text>current date and time is: </xsl:text>
    <xsl:value-of select="timezone-from-dateTime(current-dateTime())"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
The current date and time is 6:43 a.m. on Thursday, the 6th of July, 2006

The implicit timezone for the current context is: -PT4H

The timezone extracted from the current date and time is: -PT4H
```

Обратите внимание: часовой пояс по умолчанию, полученный функцией implicit-timezone(), совпадает с часовым поясом, извлеченным из текущей даты и времени. Часовой пояс по умолчанию является компонентом всех значений xs:date, xs:dateTime и xs:time, создаваемых без указания конкретного часового пояса.

[2.0] current-group()

Возвращает все объекты, включенные в текущую группу.

Синтаксис

```
item()* current-group()
```

Ввод

Нет.

Вывод

Набор объектов, входящих в текущую группу. Функция полезна только в элементе <xsl:for-each-group>. При вызове current-group() за пределами <xsl:for-each-group> возвращается пустая последовательность.

Где определяется

XSLT 2.0, раздел 14 «Grouping».

Пример

Для демонстрации работы функции current-group() используется упрощенная версия документа с адресной книгой:

```
<?xml version="1.0"?>
<!-- simple-addresses.xml -->
<addressbook>
  <address>
    <name>Mr. Chester Hasbrouck Frisby</name>
    <city>Sheboygan</city>
    <state>WI</state>
  </address>
  <address>
    <name>Mary Backstayge</name>
    <city>Skunk Haven</city>
    <state>MA</state>
  </address>
  <address>
    <name>Ms. Natalie Attired</name>
    <city>Winter Harbor</city>
    <state>ME</state>
  </address>
  <address>
    <name>Harry Backstayge</name>
    <city>Skunk Haven</city>
    <state>MA</state>
  </address>
  <address>
    <name>Mary McGoon</name>
    <city>Boylston</city>
    <state>VA</state>
  </address>
  <address>
    <name>Ms. Amanda Reckonwith</name>
    <city>Lynn</city>
    <state>MA</state>
  </address>
</addressbook>
```

Следующая таблица стилей группирует имена по элементу <state>:

```
<?xml version="1.0"?>
<!-- current-group.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>
```

```
  <xsl:template match="/">
    <xsl:text>&#xA;Here's a test of the </xsl:text>
    <xsl:text>current-group() function:&#xA;</xsl:text>
    <xsl:text>&#xA;  Customers grouped by state</xsl:text>

    <xsl:for-each-group select="//address" group-by="state">
      <xsl:sort select="state"/>
      <xsl:text>&#xA;&#xA;    Customers in </xsl:text>
      <xsl:value-of select="current-grouping-key()"/>
      <xsl:text>:</xsl:text>

      <xsl:for-each select="current-group()">
        <xsl:text>&#xA;      </xsl:text>
        <xsl:value-of select="name"/>
        <xsl:text> of </xsl:text>
        <xsl:value-of select="city"/>
      </xsl:for-each>
    </xsl:for-each-group>
  </xsl:template>

</xsl:stylesheet>
```

В этом примере люди из адресной книги группируются по штату, в котором они живут. Группы выбираются при помощи элемента `<xsl:for-each-group>`. Внутри него в элементе `<xsl:for-each>` вызывается функция `current-group()` для перебора всех объектов текущей группы.

Для нашей упрощенной адресной книги таблица стилей выдает следующий результат:

```
Here's a test of the current-group() function:

  Customers grouped by state:

    Customers in MA:
      Mary Backstayge of Skunk Haven
      Harry Backstayge of Skunk Haven
      Ms. Amanda Reckonwith of Lynn

    Customers in ME:
      Ms. Natalie Attired of Winter Harbor

    Customers in VA:
      Mary McGoon of Boylston

    Customers in WI:
      Mr. Chester Hasbrouck Frisby of Sheboygan
```

Таблицу стилей можно дополнительно усовершенствовать, используя функцию `count()` с `current-group()`. В тех штатах, в которых живет более одного клиента (`current-group()) > 1`), заголовок имеет вид `Customers in MA`. Там, где живет только один клиент (`count(current-group()) = 1`), используется заголовок вида `Our only customer in ME`.

[2.0] current-grouping-key()

Возвращает значение, используемое для выбора текущей группы.

Синтаксис

```
xs:anyAtomicType? current-grouping-key()
```

Ввод

Нет.

Вывод

Значение, используемое для выбора элементов в текущей группе. Функция `current-grouping-key()` полезна только в элементе `<xsl:for-each-group>` с атрибутом `group-by` или `group-adjacent`. При вызове `current-grouping-key()` в любом другом месте возвращается пустая последовательность.

Где определяется

XSLT 2.0, раздел 14 «Grouping».

Пример

Для демонстрации работы функции `current-grouping-key()` используется упрощенная версия документа с адресной книгой:

```
<?xml version="1.0"?>
<!-- simple-addresses.xml -->
<addressbook>
  <address>
    <name>Mr. Chester Hasbrouck Frisby</name>
    <city>Sheboygan</city>
    <state>WI</state>
  </address>
  <address>
    <name>Mary Backstayge</name>
    <city>Skunk Haven</city>
    <state>MA</state>
  </address>
  <address>
    <name>Ms. Natalie Attired</name>
    <city>Winter Harbor</city>
    <state>ME</state>
  </address>
  <address>
    <name>Harry Backstayge</name>
    <city>Skunk Haven</city>
    <state>MA</state>
  </address>
  <address>
    <name>Mary McGoon</name>
    <city>Boylston</city>
    <state>VA</state>
  </address>
  <address>
    <name>Ms. Amanda Reckonwith</name>
    <city>Lynn</city>
    <state>MA</state>
  </address>
</addressbook>
```

Следующая таблица стилей группирует имена по элементу `<state>`:

```
<?xml version="1.0"?>
<!-- current-grouping-key.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Here's a test of the </xsl:text>
    <xsl:text>current-grouping-key() function:&#xA;</xsl:text>
    <xsl:text>&#xA;  Customers grouped by state</xsl:text>

    <xsl:for-each-group select="//address" group-by="state">
      <xsl:sort select="state"/>
      <xsl:text>&#xA;&#xA;    Customers in </xsl:text>
      <xsl:value-of select="current-grouping-key()"/>
      <xsl:text>:</xsl:text>

      <xsl:for-each select="current-group()">
        <xsl:text>&#xA;      </xsl:text>
        <xsl:value-of select="name"/>
        <xsl:text> of </xsl:text>
        <xsl:value-of select="city"/>
      </xsl:for-each>
    </xsl:for-each-group>
  </xsl:template>

</xsl:stylesheet>
```

В этом примере люди из адресной книги группируются по штату, в котором они живут. В заголовке каждой группы выводится группирующее значение (штат) при помощи функции `current-grouping-key()`:

```
Here's a test of the current-grouping-key() function:

  Customers grouped by state:

    Customers in MA:
      Mary Backstayge of Skunk Haven
      Harry Backstayge of Skunk Haven
      Ms. Amanda Reckonwith of Lynn

    Customers in ME:
      Ms. Natalie Attired of Winter Harbor

    Customers in VA:
      Mary McGoon of Boylston

    Customers in WI:
      Mr. Chester Hasbrouck Frisby of Sheboygan
```

[2.0] current-time()

Возвращает объект `xs:time`, заполненный текущим временем.

Синтаксис

```
xs:time current-time()
```

Ввод

Нет.

Вывод

Значение типа `xs:time`, заполненное текущим временем системы. В возвращаемом значении устанавливается часовой пояс по умолчанию. (Для получения информации о часовом поясе по умолчанию используется функция `implicit-timezone()`.) *Функция* `current-time()` *возвращает одинаковые значения в процессе обработки таблицы стилей*. Даже если функция `current-time()` десятикратно вызывается в таблице стилей, каждый раз она возвращает одно и то же значение.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 2 «Accessors».

Пример

Следующая таблица стилей отображает текущее время. В ней используются функции `current-time()`, `format-time()` и `implicit-timezone()`.

```
<?xml version="1.0"?>
<!-- current-time.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>The current time is </xsl:text>
    <xsl:value-of
      select="format-time(current-time(),
              '[h]:[m01] [Pn]')"/>
    <xsl:text>&#xA;&#xA;The implicit timezone for the </xsl:text>
    <xsl:text>current context is: </xsl:text>
    <xsl:value-of select="implicit-timezone()"/>
    <xsl:text>&#xA;&#xA;The timezone extracted from the </xsl:text>
    <xsl:text>current date and time is: </xsl:text>
    <xsl:value-of select="timezone-from-time(current-time())"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
The current time is 6:49 a.m.

The implicit timezone for the current context is: -PT4H

The timezone extracted from the current date and time is: -PT4H
```

Обратите внимание: часовой пояс по умолчанию, полученный функцией `implicit-timezone()`, совпадает с часовым поясом, извлеченным из текущего времени. Часовой пояс по умолчанию является компонентом всех значений `xs:date`, `xs:dateTime` и `xs:time`, создаваемых без указания конкретного часового пояса.

[2.0] data()

По заданной последовательности объектов возвращает последовательность атомарных значений, представляющих объекты исходной последовательности.

Синтаксис

```
xs:anyAtomicType* data(item()*)
```

Ввод

Последовательность объектов.

Вывод

Последовательность, в которой каждый объект входной последовательности преобразуется в соответствующее атомарное значение. Если объект входной последовательности является атомарным значением, он включается в выходную последовательность в неизменном виде. Для узлов типизованные объекты преобразуются в атомарные значения и возвращаются в итоговой последовательности.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 2 «Accessors».

Пример

Следующая таблица стилей создает последовательность, состоящую из атомов и элементов, и выводит их с указанием разновидности объекта (атомарное значение или элемент). Затем таблица стилей выводит последовательность, возвращаемую функцией `data()`:

```
<?xml version="1.0"?>
<!-- data.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:datatest="http://www.oreilly.com">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <!-- Перед тестированием функции data() необходимо
         создать исходную последовательность. -->
    <xsl:variable name="testSequence" as="item()*">
      <xsl:sequence select="(3, 4, 5)"/>
      <xsl:element name="currentDate">
        <xsl:value-of select="current-date()"/>
      </xsl:element>
      <xsl:element name="currentTime">
        <xsl:value-of select="current-time()"/>
      </xsl:element>
      <xsl:element name="integerTest">
        <xsl:value-of select="xs:integer(8)"/>
      </xsl:element>
      <xsl:sequence select="('blue', 'red')"/>
      <xsl:element name="floatTest1">
```

```
        <xsl:value-of select="xs:float(3.14)"/>
      </xsl:element>
      <xsl:element name="floatTest2">
        <xsl:value-of select="xs:float(42)"/>
      </xsl:element>
      <xsl:element name="dateTest">
        <xsl:value-of select="xs:date('1995-04-21')"/>
      </xsl:element>
    </xsl:variable>

    <xsl:text>&#xA;Here is a test of the data() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  Our original sequence is:&#xA;&#xA;    </xsl:text>
    <xsl:value-of
      select="for $i in (1 to count($testSequence))
              return (datatest:print-item(
              subsequence($testSequence, $i, 1)))"
      separator="&#xA;    "/>

    <xsl:text>&#xA;&#xA;  Passing our sequence to data() </xsl:text>
    <xsl:text>gives us:&#xA;&#xA;    </xsl:text>

    <xsl:variable name="atomicSequence" as="item()*"
      select="data($testSequence)"/>
    <xsl:value-of
      select="for $i in (1 to count($atomicSequence))
              return (datatest:print-item(
              subsequence($atomicSequence, $i, 1)))"
      separator="&#xA;    "/>
  </xsl:template>

  <!-- Для заданного объекта item() функция возвращает строку -->
  <!-- с описанием этого объекта. -->
  <xsl:function name="datatest:print-item" as="xs:string">
    <xsl:param name="item" as="item()"/>
    <xsl:choose>
      <xsl:when test="$item instance of element()">
        <xsl:value-of
          select="concat('Element:         &lt;', name($item), '&gt;', $item,
                  '&lt;/', name($item), '&gt;')"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of
          select="concat('Atomic value:    ', $item)"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:function>

</xsl:stylesheet>
```

Результат:

```
Here is a test of the data() function:

  Our original sequence is:

    Atomic value:    3
    Atomic value:    4
    Atomic value:    5
```

```
  Element:          <currentDate>2008-03-04-05:00</currentDate>
  Element:          <currentTime>23:58:38.609-05:00</currentTime>
  Element:          <integerTest>8</integerTest>
  Atomic value:     blue
  Atomic value:     red
  Element:          <floatTest1>3.14</floatTest1>
  Element:          <floatTest2>42</floatTest2>
  Element:          <dateTest>1995-04-21</dateTest>

Passing our sequence to data() gives us:

  Atomic value:     3
  Atomic value:     4
  Atomic value:     5
  Atomic value:     2008-03-04-05:00
  Atomic value:     23:58:38.609-05:00
  Atomic value:     8
  Atomic value:     blue
  Atomic value:     red
  Atomic value:     3.14
  Atomic value:     42
  Atomic value:     1995-04-21
```

Тестовая последовательность содержит 11 объектов, 5 из которых являются атомарными значениями, а еще 6 – элементами. Каждый объект поочередно передается функции `datatest:print-item()`, проверяющей тип объекта. Для типа `element()` функция возвращает строку с открывающим тегом элемента, его значением и закрывающим тегом. Для атомов функция просто возвращает значение атома. Все строки, возвращаемые функцией `datatest:print-item()`, помечаются префиксом `Element:` или `Atom:`; префикс показывает, является данный элемент элементом или атомом. Таким образом выводится содержимое исходной последовательности.

После вывода всех атомов и элементов исходной последовательности функция `data()` преобразует исходную последовательность в последовательность атомарных значений. Вызов функции `datatest:print-item()` для каждого объекта новой последовательности показывает, что новая последовательность состоит из одних атомов.

В таблице стилей следующее выражение XPath вызывает функцию `datatest:print-item()` для каждого объекта последовательности:

```
select="for $i in (1 to count($testSequence))
        return (datatest:print-item(
        subsequence($testSequence, $i, 1)))"
```

Выражение использует новую конструкцию XPath для перебора объектов последовательности (`count($testSequence)`). Для каждого объекта возвращается `xs:string`, полученное от нашей функции `datatest:print-item()`. Для сохранения структуры таблицы стилей в шаблоне `match="/"` мы используем относительно сложные выражения XPath и создаем функцию для выполнения многократно повторяемой операции вывода каждого объекта.

При желании можно создать выражение XPath, которое будет выполнять всю работу функции. Этот код выглядит примерно так:

```
<!-- data2.xsl -->
...
    <xsl:for-each select="$testSequence">
      <xsl:value-of
```

```
          select="if (. instance of node())
                  then concat('Element:         &lt;', name(.),
                              '&gt;', ., '&lt;/', name(.), '&gt;')
                  else concat('Atomic value:    ', .)"/>
        <xsl:text>&#xA;    </xsl:text>
      </xsl:for-each>
...
      <xsl:variable name="atomicSequence" as="item()*"
        select="data($testSequence)"/>
      <xsl:for-each select="$atomicSequence">
        <xsl:value-of
          select="if (. instance of item())
                  then concat('Atomic value:    ', .)
                  else concat('Element:         &lt;', name(.),
                              '&gt;', ., '&lt;/', name(.), '&gt;')"/>
        <xsl:text>&#xA;    </xsl:text>
      </xsl:for-each>
...
```

Такое решение работает, но логику `if-then-else` придется дублировать всюду, где потребуется вывести последовательность. Для выполнения однообразной работы гораздо лучше использовать элемент `<xsl:function>`. Кроме того, процессор XSLT может выдавать предупреждения о безопасности типов при вызовах функции `name()`. Если изменить обработку `$atomicSequence` так, чтобы она начиналась с `. instance of node()`, процессор выдаст статическую ошибку, и таблица стилей работать не будет. Для предотвращения подобных проблем проще оформить этот код в виде функции.

[2.0] dateTime()

Специальная функция-конструктор для создания значений `xs:dateTime` по значениям `xs:date` и `xs:time`.

Синтаксис

```
xs:dateTime? dateTime(xs:date?, xs:time?)
```

Ввод

Значение `xs:date` и значение `xs:time`.

Вывод

Новое значение `xs:dateTime`, созданное на основании двух входных значений. Если хотя бы одно входное значение является пустой последовательностью, `xs:dateTime()` возвращает пустую последовательность. Часовой пояс результата `xs:dateTime` вычисляется по следующим правилам:

- Если ни один аргумент не содержит часового пояса, то и результат не содержит часового пояса.
- Если часовой пояс задан только в одном аргументе или в обоих аргументах заданы одинаковые часовые пояса, результатом будет этот часовой пояс.
- Если два аргумента содержат разные часовые пояса, процессор XSLT выдает ошибку.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 5 «Constructor Functions».

Пример

Таблица стилей использует значения, полученные при помощи функций `current-date()` и `current-time()`, для создания нового значения `xs:dateTime`:

```
<?xml version="1.0"?>
<!-- datetime.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Creating an xs:dateTime with an </xsl:text>
    <xsl:text>xs:date and xs:time:</xsl:text>
    <xsl:variable name="currentDate" as="xs:date" select="current-date()"/>
    <xsl:variable name="currentTime" as="xs:time" select="current-time()"/>
    <xsl:text>&#xA;&#xA;  The current date is: </xsl:text>
    <xsl:value-of select="$currentDate"/>
    <xsl:text>&#xA;&#xA;  The current time is: </xsl:text>
    <xsl:value-of select="$currentTime"/>

    <xsl:variable name="currentDateTime"
      select="dateTime($currentDate, $currentTime)"/>
      <xsl:text>&#xA;&#xA;  The new xs:dateTime is: </xsl:text>
      <xsl:value-of select="$currentDateTime"/>
      <xsl:text>&#xA;&#xA;  The new xs:dateTime value can be </xsl:text>
      <xsl:text>written as &#xA;    </xsl:text>
      <xsl:value-of
      select="format-dateTime($currentDateTime,
              '[h]:[m01] [Pn] on [FNn], the [D1o] of [MNn], [Y0001]')"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Creating an xs:dateTime with an xs:date and xs:time:

  The current date is: 2006-11-16-05:00

  The current time is: 03:48:09.888-05:00

  The new xs:dateTime is: 2006-11-16T03:48:09.888-05:00

  The new xs:dateTime value can be written as
    3:48 a.m. on Thursday, the 16th of November, 2006
```

[2.0] day-from-date()

Возвращает компонент дня для полученного значения `xs:date`.

Синтаксис

```
xs:integer? day-from-date(xs:date?)
```

Ввод

Значение `xs:date`.

Вывод

Значение `xs:integer`, представляющее компонент дня для заданного значения `xs:date`. Если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Для демонстрации работы функции `day-from-date()` используется следующая таблица стилей:

```
<?xml version="1.0"?>
<!-- day-from-date.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the day from an xs:date:</xsl:text>
    <xsl:variable name="currentDate" as="xs:date" select="current-date()"/>
    <xsl:text>&#xA;&#xA;The current date is: </xsl:text>
    <xsl:value-of select="$currentDate"/>

    <xsl:text>&#xA;&#xA;  The current day: </xsl:text>
    <xsl:value-of select="day-from-date($currentDate)"/>
    <xsl:text>&#xA;    In words: </xsl:text>
    <xsl:value-of select="format-date($currentDate, '[DWw]')"/>
    <xsl:text>&#xA;    In German: </xsl:text>
    <xsl:value-of select="format-date($currentDate, '[Dw]', 'de', (), ())"/>
    <xsl:text>&#xA;    It's the </xsl:text>
    <xsl:value-of select="format-date($currentDate, '[dwo]')"/>
    <xsl:text> day of the year.</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей создает следующий результат:

```
Extracting the day from an xs:date:

The current date is: 2006-11-16-05:00

  The current day: 16
    In words: Sixteen
    In German: sechszehn
    It's the three hundred and twentieth day of the year.
```

Обратите внимание: одни результаты генерируются функцией `day-from-date()`, а другие — функцией `format-date()` с форматной строкой, выбирающей только компонент дня из значения `xs:date`.

См. также

Определения функций *[2.0]* `format-date()`, *[2.0]* `month-from-date()`, *[2.0]* `timezone-from-date()` и *[2.0]* `year-from-date()`.

[2.0] day-from-dateTime()

Возвращает компонент дня для полученного значения `xs:dateTime`.

Синтаксис

```
xs:integer? day-from-dateTime(xs:dateTime?)
```

Ввод

Значение `xs:dateTime`.

Вывод

Значение `xs:integer`, представляющее компонент дня для заданного значения `xs:dateTime`. Если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Для демонстрации работы функции `day-from-dateTime()` используется следующая таблица стилей:

```
<?xml version="1.0"?>
<!-- day-from-datetime.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

 <xsl:template match="/">
    <xsl:text>&#xA;Extracting the day from an xs:dateTime:</xsl:text>
    <xsl:variable name="currentDateTime" as="xs:dateTime"
      select="current-dateTime()"/>
    <xsl:text>&#xA;&#xA;The current date and time is: </xsl:text>
    <xsl:value-of select="$currentDateTime"/>

    <xsl:text>&#xA;&#xA;  The current day: </xsl:text>
    <xsl:value-of select="day-from-dateTime($currentDateTime)"/>
    <xsl:text>&#xA;    In words: </xsl:text>
    <xsl:value-of select="format-dateTime($currentDateTime, '[DWw]')"/>
    <xsl:text>&#xA;    In German words: </xsl:text>
    <xsl:value-of
      select="format-dateTime($currentDateTime, '[Dw]', 'de', (), ())"/>
    <xsl:text>&#xA;    It's the </xsl:text>
    <xsl:value-of select="format-dateTime($currentDateTime, '[dwo]')"/>
    <xsl:text> day of the year.</xsl:text>
  </xsl:template>
```

```
</xsl:stylesheet>
```

Результат:

```
Extracting the day from an xs:dateTime:

The current date and time is: 2006-11-16T04:04:46.811-05:00

  The current day: 16
    In words: Sixteen
    In German words: sechszehn
    It's the three hundred and twentieth day of the year.
```

Обратите внимание: одни результаты генерируются функцией `day-from-dateTime()`, а другие – функцией `format-dateTime()` с форматной строкой, выбирающей только компонент дня из значения `xs:dateTime`.

См. также

Определения функций ***[2.0]*** `format-dateTime()`, ***[2.0]*** `hours-from-dateTime()`, ***[2.0]*** `minutes-from-dateTime()`, ***[2.0]*** `month-from-dateTime()`, ***[2.0]*** `seconds-from-dateTime()`, ***[2.0]*** `timezone-from-dateTime()` и ***[2.0]*** `year-from-dateTime()`.

[2.0] days-from-duration()

Возвращает количество дней в заданном временном промежутке `xs:duration`.

Синтаксис

```
xs:integer? days-from-duration(xs:duration?)
```

Ввод

Значение `xs:duration`.

Вывод

Значение `xs:integer`, представляющее количество дней в заданном промежутке `xs:duration`. Учтите, что для `xs:yearMonthDuration` эта функция всегда возвращает `0`, потому что в значении `xs:yearMonthDuration` отсутствует компонент дней. Кроме того, если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Следующая таблица стилей демонстрирует использование функции `days-from-duration()` для всех трех типов временных промежутков:

```
<?xml version="1.0"?>
<!-- days-from-duration.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the days component from durations:</xsl:text>

    <xsl:variable name="sampleDuration" as="xs:duration"
      select="xs:duration('P3Y8M2DT4H23M12.2S')"/>
    <xsl:variable name="sampleYearMonthDuration" as="xs:yearMonthDuration"
      select="xs:yearMonthDuration('P3Y8M')"/>
    <xsl:variable name="sampleDayTimeDuration" as="xs:dayTimeDuration"
      select="xs:dayTimeDuration('P2DT4H23M12.2S')"/>

    <xsl:text>&#xA;&#xA;  A sample xs:duration: </xsl:text>
    <xsl:value-of select="$sampleDuration"/>
    <xsl:text>&#xA;    The days component of this duration is </xsl:text>
    <xsl:value-of select="days-from-duration($sampleDuration)"/>
    <xsl:text>.</xsl:text>

    <xsl:text>&#xA;&#xA;  A sample xs:yearMonthDuration: </xsl:text>
    <xsl:value-of select="$sampleYearMonthDuration"/>
    <xsl:text>&#xA;    The days component of this duration is </xsl:text>
    <xsl:value-of select="days-from-duration($sampleYearMonthDuration)"/>
    <xsl:text>.</xsl:text>

    <xsl:text>&#xA;&#xA;  A sample xs:dayTimeDuration: </xsl:text>
    <xsl:value-of select="$sampleDayTimeDuration"/>
    <xsl:text>&#xA;    The days component of this duration is </xsl:text>
    <xsl:value-of select="days-from-duration($sampleDayTimeDuration)"/>
    <xsl:text>.</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Extracting the days component from durations:

  A sample xs:duration: P3Y8M2DT4H23M12.2S
    The days component of this duration is 2.

  A sample xs:yearMonthDuration: P3Y8M
    The days component of this duration is 0.

  A sample xs:dayTimeDuration: P2DT4H23M12.2S
    The days component of this duration is 2.
```

Как упоминалось ранее, при вызове `days-from-duration()` **для** `xs:yearMonthDuration` **возвращается** `0`**.**

См. также

Определения функций ***[2.0]*** `hours-from-duration()`**,** ***[2.0]*** `minutes-from-duration()`**,** ***[2.0]*** `months-from-duration()`**,** ***[2.0]*** `seconds-from-duration()` **и** ***[2.0]*** `years-from-duration()`**.**

[2.0] deep-equal()

Сравнивает две последовательности объектов и проверяет, равны ли эти последовательности вместе со всеми потомками.

Синтаксис

```
xs:boolean deep-equal(item()*, item()*)
xs:boolean deep-equal(item()*, item()*, $collation as xs:string)
```

Ввод

Две последовательности объектов. Необязательный третий аргумент определяет схему упорядочения, которая должна использоваться при сравнении строковых значений. (Упорядочение не используется при сравнении имен узлов, так что узлы `<strasse>` и `<straße>` никогда не равны, даже если схема упорядочения для немецкого языка считает эти два строковых значения равными.)

Вывод

`true`, если последовательности находятся в отношении *глубокого равенства*; `false` в противном случае. С учетом разнообразия типов объектов в XSLT 2.0 правила глубокого равенства двух последовательностей получаются относительно сложными:

- Если два значения являются пустыми последовательностями, `deep-equal()` возвращает `true`.
- Если два значения являются последовательностями разной длины (`count($value1) != count($value2)`), `deep-equal()` возвращает `false`.
- Если два значения являются узлами разных типов (например, элемент и атрибут), `deep-equal()` возвращает `false`.
- Если два значения являются узлами документов, они считаются равными только в том случае, если `$value1/(*|text())` находится в отношении глубокого равенства с `$value2/(*|text())`.
- Если два значения являются узлами элементов, они должны иметь одинаковые имена и одинаковое количество атрибутов. Кроме того, атрибуты должны иметь одинаковые значения, а все дочерние узлы должны находиться в отношениях глубокого равенства.
- Для остальных типов узлов (атрибуты, инструкции по обработке, комментарии, текст) действуют весьма простые правила. Для атрибутов и инструкций по обработке узлы должны иметь одинаковые имена и одинаковые значения. Для комментариев и текста их строковые значения должны быть равны.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.3 «Equals, Union, Intersection and Except».

Пример

Для тестирования функции `deep-equal()` будет использоваться документ с информацией о продажах шоколада:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
```

```
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Следующая таблица стилей сравнивает несколько значений. Обратите внимание: функция `deep-equal()` сравнивает как атомарные значения, так и узлы.

```
<?xml version="1.0"?>
<!-- deep-equal.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:variable name="testTree" as="node()*">
    <ora:report month="8" year="2006"
      xmlns:ora="http://www.oreilly.com">
      <ora:title>Chocolate bar sales</ora:title>
      <ora:brand>
        <ora:name>Lindt</ora:name>
        <ora:units>27408</ora:units>
      </ora:brand>
    </ora:report>

    <dw:report month="8" year="2006"
      xmlns:dw="http://www.oreilly.com">
      <dw:title>Chocolate bar sales</dw:title>
      <dw:brand>
        <dw:name>Lindt</dw:name>
        <dw:units>27408</dw:units>
      </dw:brand>
    </dw:report>

    <report month="8" year="2006">
      <title>Chocolate bar sales</title>
      <brand>
        <name>Lindt</name>
        <units>27408</units>
      </brand>
    </report>

    <address>
      <company>IBM Boeblingen</company>
      <street>Schoenaicherstrasse 220</street>
```

```
  </address>

  <address>
    <company>IBM B&#xF6;blingen</company>
    <street>Sch&#xF6;naicherstra&#xDF;e 220</street>
  </address>
</xsl:variable>

<xsl:variable name="sequence1" as="item()*">
  <xsl:sequence select="(3, 4, 5)"/>
</xsl:variable>

<xsl:variable name="sequence2" as="item()*">
  <xsl:sequence select="(3, 5, 4)"/>
</xsl:variable>

<xsl:variable name="sequence3" as="item()*">
  <xsl:sequence select="(3, 4, 5)"/>
</xsl:variable>

<xsl:template match="/">
  <xsl:text>&#xA;Tests of the deep-equal() function:</xsl:text>

  <xsl:text>&#xA;&#xA;  $sequence1 = (</xsl:text>
  <xsl:value-of select="$sequence1" separator=", "/>
  <xsl:text>)</xsl:text>
  <xsl:text>&#xA;  $sequence2 = (</xsl:text>
  <xsl:value-of select="$sequence2" separator=", "/>
  <xsl:text>)</xsl:text>
  <xsl:text>&#xA;  $sequence3 = (</xsl:text>
  <xsl:value-of select="$sequence3" separator=", "/>
  <xsl:text>)</xsl:text>

  <xsl:text>&#xA;&#xA;  deep-equal(</xsl:text>
  <xsl:text>$sequence1, $sequence2) = </xsl:text>
  <xsl:value-of select="deep-equal($sequence1, $sequence2)"/>

  <xsl:text>&#xA;&#xA;  deep-equal(</xsl:text>
  <xsl:text>$sequence1, $sequence3) = </xsl:text>
  <xsl:value-of select="deep-equal($sequence1, $sequence3)"/>

  <xsl:text>&#xA;&#xA;  deep-equal(</xsl:text>
  <xsl:text>subsequence($sequence1, 3, 1), &#xA;</xsl:text>
  <xsl:text>             subsequence($sequence2, 2, 1) = </xsl:text>
  <xsl:value-of
    select="deep-equal(subsequence($sequence1, 3, 1),
            subsequence($sequence2, 2, 1))"/>

  <xsl:text>&#xA;&#xA;  Comparing the first two </xsl:text>
  <xsl:text>subtrees in $testTree:</xsl:text>
  <xsl:text>&#xA;    deep-equal(subsequence($testTree, 1, 1),&#xA;</xsl:text>
  <xsl:text>               subsequence($testTree, 2, 1)) = </xsl:text>
  <xsl:value-of
    select="deep-equal(subsequence($testTree, 1, 1),
            subsequence($testTree, 2, 1))"/>

  <xsl:text>&#xA;&#xA;  Comparing part of our input </xsl:text>
  <xsl:text>document to &#xA;      part of the third subtree </xsl:text>
  <xsl:text>in $testTree:</xsl:text>
  <xsl:text>&#xA;    deep-equal(/report/brand[1]/units,$#xA;</xsl:text>
  <xsl:text>               subsequence($testTree, 3, 1)</xsl:text>
```

```
    <xsl:text>/brand[1]/units) = </xsl:text>
    <xsl:value-of
      select="deep-equal(/report/brand[1]/units,
              subsequence($testTree, 3, 1)/brand[1]/units)"/>

    <xsl:text>&#xA;&#xA;  Comparing two German addresses:</xsl:text>
    <xsl:text>&#xA;    deep-equal(subsequence($testTree, 4, 1),&#xA;</xsl:text>
    <xsl:text>               subsequence($testTree, 5, 1)) = </xsl:text>
    <xsl:value-of
      select="deep-equal(subsequence($testTree, 4, 1),
              subsequence($testTree, 5, 1))"/>

    <xsl:text>&#xA;&#xA;  Comparing two German addresses </xsl:text>
    <xsl:text>using German collation: </xsl:text>
    <xsl:text>&#xA;    deep-equal(subsequence($testTree, 4, 1),&#xA;</xsl:text>
    <xsl:text>               subsequence($testTree, 5, 1), &#xA;</xsl:text>
    <xsl:text>               $GermanCollation) = </xsl:text>
    <xsl:value-of
      select="deep-equal(subsequence($testTree, 4, 1),
              subsequence($testTree, 5, 1),
              concat('http://saxon.sf.net/collation?',
                     'class=com.oreilly.xslt.GermanCollation;'))"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the deep-equal() function:

  $sequence1 = (3, 4, 5)
  $sequence2 = (3, 5, 4)
  $sequence3 = (3, 4, 5)

  deep-equal($sequence1, $sequence2) = false

  deep-equal($sequence1, $sequence3) = true

  deep-equal(subsequence($sequence1, 3, 1),
             subsequence($sequence2, 2, 1) = true

  Comparing the first two subtrees in $testTree:
    deep-equal(subsequence($testTree, 1, 1),
               subsequence($testTree, 2, 1)) = true

  Comparing part of our input document to
      part of the third subtree in $testTree:
    deep-equal(/report/brand[1]/units,
               subsequence($testTree, 3, 1)/brand[1]/units) = true

  Comparing two German addresses:
    deep-equal(subsequence($testTree, 4, 1),
               subsequence($testTree, 5, 1)) = false

  Comparing two German addresses using German collation:
    deep-equal(subsequence($testTree, 4, 1),
               subsequence($testTree, 5, 1),
               $GermanCollation) = true
```

Все начинается с создания переменной, содержащей три корневых узла. Первые два узла идентичны и отличаются только префиксами пространств имен. Третий корневой узел `$testTree` напоминает часть документа с данными о продажах шоколада.

Также создаются три последовательности атомарных значений. Последовательности атомов сравниваются друг с другом, а один объект из $sequence1 сравнивается с одним объектом из $sequence2.

Сравнение первых двух корневых узлов в $testTree показывает, что они фактически находятся в отношении глубокого равенства друг с другом, потому что префикс пространства имен роли не играет. Оба префикса отображаются на один URI-адрес, поэтому два корневых узла равны. Если два узла имеют одинаковые префиксы, но эти одинаковые префиксы соответствуют разным URI-адресам, такие два узла не пройдут проверку на глубокое равенство.

Затем элемент <units> из первого элемента <brand> в третьем корневом узле $testTree сравнивается с аналогичным элементом входного документа. Эти два узла находятся в отношении глубокого равенства.

Наконец, в $testTree добавляются два элемента <address>. При использовании схемы упорядочения по умолчанию текстовые значения IBM Boeblingen и IBM Böblingen различны, как и значения Schoenaicherstrasse 220 и Schönaicherstraße 220. При использовании упорядочения для немецкого языка эти строки равны, поэтому функция deep-equal() возвращает true. (Чтобы листинг поместился на странице, мы объединили две половины URI-адреса схемы упорядочения Saxon при помощи функции concat().) За дополнительной информацией обращайтесь к разделу «Функция document() и сортировка» главы 8.

[2.0] default-collation()

Возвращает строку, представляющую схему упорядочения по умолчанию.

Синтаксис

```
xs:string default-collation()
```

Ввод

Нет.

Вывод

Значение xs:string, представляющее схему упорядочения по умолчанию.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 16 «Context Functions».

Пример

Следующая таблица стилей просто выводит схему упорядочения по умолчанию:

```
<?xml version="1.0"?>
<!-- default-collation.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;The default collation is: &#xA;  </xsl:text>
    <xsl:value-of select="default-collation()"/>
```

```
    </xsl:template>

</xsl:stylesheet>
```

Результат:

```
The default collation is:
  http://www.w3.org/2005/xpath-functions/collation/codepoint
```

В соответствии со спецификацией XQuery 1.0 and XPath 2.0 Functions and Operators по умолчанию используется схема упорядочения кодов символов (кодовых пунктов) Юникода. Эта схема упорядочения связывается с URI-адресом `http://www.w3.org/2005/xpath-functions/collation/codepoint` и должна поддерживаться всеми реализациями XPath 2.0 и XQuery 1.0. Реализации могут поддерживать и другие схемы упорядочения, и каждый процессор может определять собственные механизмы связывания URI-адресов с конкретной схемой упорядочения. Кроме того, каждый процессор может определять собственную схему упорядочения по умолчанию.

Последнее замечание: схема упорядочения по умолчанию может переопределяться в функциях, указывающих последовательность упорядочения, таких как `compare()`, `deep-equal()`, `ends-with()` и `max()`.

[2.0] distinct-values()

Для заданной последовательности возвращает новую последовательность, содержащую один экземпляр каждого уникального значения в исходной последовательности.

Синтаксис

```
xs:anyAtomicType* distinct-values(xs:anyAtomicType*)
xs:anyAtomicType* distinct-values(xs:anyAtomicType*, $collation as xs:string)
```

Ввод

Последовательность атомарных значений. Функция `distinct-values()` также имеет необязательный аргумент, задающий алгоритм упорядочения. Если этот аргумент присутствует, решение о том, отличаются ли два значения друг от друга, принимается по этому алгоритму.

Вывод

Последовательность, содержащая только одну копию каждого атомарного значения в исходной последовательности. При использовании функции следует учитывать некоторые обстоятельства:

- Порядок следования уникальных значений в последовательности, возвращаемой функцией `distinct-values()`, зависит от реализации. Выбор значения, возвращаемого из группы одинаковых значений, также зависит от реализации.
- Если входная последовательность является пустой последовательностью, функция возвращает пустую последовательность. Передача пустой последовательности `distinct-values()` не приводит к ошибке.
- При сравнении значений `xs:float` и `xs:double` положительный и отрицательный нули считаются равными. Кроме того, если в исходной последовательности присутствуют значения `NaN` («нечисло»), в итоговую последовательность включается только одно из этих значений.

- **При сравнении значений** `xs:date`, `xs:dateTime` **и** `xs:time`, **если эти значения не содержат компонента часового пояса, по умолчанию используется часовой пояс, возвращаемый функцией** `implicit-timezone()`. **Также учтите, что значения** `xs:date`, `xs:dateTime` **и** `xs:time` **могут быть равными при различающихся компонентах часового пояса (например, полдень в одном часовом поясе может быть равен 9:00 часам в другом часовом поясе).**

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.1 «General Functions and Operators on Sequences».

Пример

Для демонстрации работы функции `distinct-values()` будет использоваться упрощенная версия адресной книги:

```
<?xml version="1.0"?>
<!-- simple-addresses.xml -->
<addressbook>
  <address>
    <name>Mr. Chester Hasbrouck Frisby</name>
    <city>Sheboygan</city>
    <state>WI</state>
  </address>
  <address>
    <name>Mary Backstayge</name>
    <city>Skunk Haven</city>
    <state>MA</state>
  </address>
  <address>
    <name>Ms. Natalie Attired</name>
    <city>Winter Harbor</city>
    <state>ME</state>
  </address>
  <address>
    <name>Harry Backstayge</name>
    <city>Skunk Haven</city>
    <state>MA</state>
  </address>
  <address>
    <name>Mary McGoon</name>
    <city>Boylston</city>
    <state>VA</state>
  </address>
  <address>
    <name>Ms. Amanda Reckonwith</name>
    <city>Lynn</city>
    <state>MA</state>
  </address>
</addressbook>
```

Простая таблица стилей для вывода уникальных значений элементов `<state>` из исходного документа XML:

```
<?xml version="1.0"?>
<!-- distinct-values1.xsl -->
```

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Here is a test of the distinct-values() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  The states in the source </xsl:text>
    <xsl:text>document are:&#xA;    </xsl:text>
    <xsl:value-of
      select="/addressbook/address/state"
      separator=", "/>

    <xsl:text>&#xA;&#xA;  The unique states </xsl:text>
    <xsl:text>is:&#xA;    </xsl:text>
    <xsl:value-of
      select="distinct-values(/addressbook/address/state)"
      separator=", "/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here is a test of the distinct-values() function:

  The states in the source document are:
    WI, MA, ME, MA, VA, MA

  The unique states is:
    WI, MA, ME, VA
```

В этом примере функция `distinct-values()` возвращает значения в порядке документа, тогда как другие процессоры XSLT могут возвращать значения в порядке сортировки или в каком-то ином порядке.

Следующая таблица стилей демонстрирует особые случаи сравнения, упоминавшиеся ранее:

```
<?xml version="1.0"?>
<!-- distinct-values2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Here are some tests of the </xsl:text>
    <xsl:text>distinct-values() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  A sequence of xs:time values:</xsl:text>
    <xsl:variable name="timeSequence" as="xs:time*">
      <xsl:sequence
        select="(xs:time('00:43:00-06:00'),
                 xs:time('01:43:00-05:00'),
                 xs:time('02:43:00'),
                 xs:time('07:35:00'),
```

```
                current-time())"/>
    </xsl:variable>

    <xsl:text>&#xA;&#xA;    </xsl:text>
    <xsl:value-of
      select="distinct-values($timeSequence)"
      separator="&#xA;    "/>

    <xsl:text>&#xA;&#xA;  A sequence of integer values:</xsl:text>
    <xsl:variable name="numberSequence" as="xs:integer*">
      <xsl:sequence select="(3, 8, (2 + 1), -0, 0)"/>
    </xsl:variable>

    <xsl:text>&#xA;    </xsl:text>
    <xsl:value-of
      select="distinct-values($numberSequence)"
      separator=", "/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Here are some tests of the distinct-values() function:

  A sequence of xs:time values:

    00:43:00-06:00
    07:35:00
    06:23:05.19-04:00

  A sequence of integer values:
    3, 8, 0
```

В первой последовательности сравниваются пять разных значений `xs:time`**. Первые три в действительности равны; они представляют один момент времени в трех разных часовых поясах. Обратите внимание на использование часового пояса по умолчанию в третьем значении** `xs:time`**; при обработке этого примера на компьютере, на котором установлен часовой пояс, отличный от GMT-4, вы получите другой результат. Четвертое и пятое значения различны, поэтому выходная последовательность содержит три значения. В приведенном примере использовался Saxon; для дубликатов этот процессор возвращает первое найденное значение.**

Вторая последовательность содержит несколько числовых значений. Значения `3` **и** `(2 + 1)` **идентичны, как** `-0` **и** `0`**. Обратите внимание: на этот раз в выходную последовательность включается второе нулевое значение вместо первого (Saxon нормализует отрицательный нуль до нуля, хотя это поведение зависит от реализации).**

В последнем примере используется схема упорядочения. Таблица стилей:

```
<?xml version="1.0"?>
<!-- distinct-values3.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A final test of the </xsl:text>
    <xsl:text>distinct-values() function:&#xA;&#xA;</xsl:text>
```

```
    <xsl:variable name="testStrings" as="xs:string*">
      <xsl:sequence
        select="'street', 'Strasse', 'Stra&#xDF;e'"/>
    </xsl:variable>

    <xsl:text>  The test sequence: &#xA;    </xsl:text>
    <xsl:value-of select="$testStrings" separator=", "/>
    <xsl:text>&#xA;  distinct-values($testStrings):</xsl:text>
    <xsl:text>&#xA;    </xsl:text>
    <xsl:value-of select="distinct-values($testStrings)"
      separator=", "/>

    <xsl:text>&#xA;  distinct-values($testStrings, </xsl:text>
    <xsl:text>[German collation]):&#xA;    </xsl:text>
    <xsl:value-of
      select="distinct-values($testStrings,
              concat('http://saxon.sf.net/collation?',
                     'class=com.oreilly.xslt.GermanCollation;'))"
      separator=", "/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
A final test of the distinct-values() function:

  The test sequence:
    street, Strasse, Straße
  distinct-values($testStrings):
    street, Strasse, Straße
  distinct-values($testStrings, [German collation]):
    street, Strasse
```

В немецком языке символ «длинного s» (ß) эквивалентен `ss`. Схема упорядочения по умолчанию игнорирует это обстоятельство, но в нашей специализированной схеме оно учитывается. (Чтобы листинг поместился на странице, мы объединили две половины URI-адреса схемы упорядочения Saxon при помощи функции `concat()`.) За дополнительной информацией обращайтесь к разделу «Функция document() и сортировка» главы 8.

[2.0] doc()

Для заданного URI-адреса возвращает узел документа, представляющий его содержимое.

Синтаксис

```
document-node()? doc($uri as xs:string?)
```

Ввод

URI-адрес (в формате, определяемом реализацией).

Вывод

Узел документа, представляющий содержимое запрашиваемого URI-адреса.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.5 «Functions and Operators that Generate Sequences».

Пример

В следующем примере используется поддержка простых имен файлов в URI-адресах, реализованная в Saxon. Функция `doc()` вызывается для загрузки следующего документа XML:

```
<?xml version="1.0"?>
<!-- polist.xml -->
<collection>
  <doc href="po38292.xml"/>
  <doc href="po38293.xml"/>
  <doc href="po38294.xml"/>
  <doc href="po38295.xml"/>
</collection>
```

Таблица стилей:

```
<?xml version="1.0"?>
<!-- doc.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the doc() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  Here are all the purchase orders </xsl:text>
    <xsl:text>listed in polist.xml:</xsl:text>
    <xsl:for-each select="doc('polist.xml')/collection/doc">
      <xsl:text>&#xA;    </xsl:text>
      <xsl:value-of select="@href"/>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Функция `doc()` используется для загрузки файла `polist.xml`, после чего содержимое загруженного документа обрабатывается элементом `<xsl:for-each>`. Результат:

```
A test of the doc() function:

  Here are all the purchase orders listed in polist.xml:
    po38292.xml
    po38293.xml
    po38294.xml
    po38295.xml
```

Обратите внимание на использование функции `doc()` в выражении XPath. Мы перебираем все элементы `<doc>` исходного файла с использованием корня документа, возвращаемого функцией `doc()`. Функция `doc()` могла бы также использоваться для открытия всех документов, упоминаемых в файле *polist.xml*.

См. также

Функции *[2.0]* `collection()`, *[2.0]* `doc-available()` и `document()`; раздел «[2.0] Функции doc() и doc-available()» главы 8.

[2.0] doc-available()

Проверяет, доступен ли заданный узел документа.

Синтаксис

```
xs:boolean doc-available($uri as xs:string)
```

Ввод

URI-адрес документа.

Вывод

Функция возвращает `true`, если функция `doc()` может успешно загрузить запрашиваемый URI-адрес. Если аргумент не может быть преобразован в `xs:anyURI`, процессор XSLT выдает ошибку. В остальных случаях `doc-available()` возвращает `false`. Формат URI-адреса определяется реализацией.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.5 «Functions and Operators that Generate Sequences».

Пример

Следующая короткая таблица стилей проверяет доступность двух документов. В ней используется поддержка простых имен файлов в URI-адресах, реализованная в Saxon:

```
<?xml version="1.0"?>
<!-- doc-available.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the doc-available() function:&#xA;</xsl:text>
    <xsl:text>&#xA;  doc-available('polist.xml') = </xsl:text>
    <xsl:value-of select="doc-available('polist.xml')"/>

    <xsl:text>&#xA;&#xA;  doc-available('polist2.xml') = </xsl:text>
    <xsl:value-of select="doc-available('polist2.xml')"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the doc-available() function:

  doc-available('polist.xml') = true

  doc-available('polist2.xml') = false
```

Файл *polist.xml* существует в текущем каталоге и может быть загружен Saxon; файл *polist2.xml* в текущем каталоге не существует.

Многие детали, касающиеся функции `doc-available()`, определяются реализацией. За информацией о реализации функции `doc-available()` и о поддерживаемых форматах URI-адресов обращайтесь к документации своего процессора XSLT.

document()

Позволяет обработать несколько исходных документов в одной таблице стилей. Эта чрезвычайно мощная и гибкая функция подробно описана в главе 7, поэтому здесь мы ограничимся ее кратким обзором.

Синтаксис

```
[1.0] node-set document(object, node-set?)
[2.0] node()* document(item()*)
[2.0] node()* document(item()*, node())
```

Ввод

Функция `document()` чаще всего вызывается со строковым аргументом; эта строка интерпретируется как URI-адрес, процессор XSLT пытается открыть его и разобрать документ. Если строка пуста (вызов имеет вид `document('')`), функция `document()` разбирает саму таблицу стилей. За подробными сведениями о параметрах функции `document()` обращайтесь к разделу «Группировка узлов» главы 7.

[2.0] В XSLT 2.0 функция `document()` может также получать второй аргумент – узел, используемый для поиска свойства базового URI-адреса запрашиваемых документов. Базовый URI-адрес узла объединяется с именами ресурсов в первом аргументе для формирования полного URI-адреса.

Вывод

[1.0] Набор узлов или *[2.0]* последовательность с узлами, определяемыми переданным аргументом. Подробности приводятся в главе 7, так что мы не будем их здесь повторять.

Где определяется

[1.0] XSLT, раздел 12.1 «Multiple Source Documents».

[2.0] XSLT, раздел 16.1 «Multiple Source Documents».

Пример

В следующем примере функция `document()` используется с пустой строкой для реализации таблицы поиска. Документ XML:

```
<?xml version="1.0"?>
<!-- polist.xml -->
<collection>
  <doc href="po38292.xml"/>
  <doc href="po38293.xml"/>
  <doc href="po38294.xml"/>
  <doc href="po38295.xml"/>
</collection>
```

Функция `document()` применяется к каждому атрибуту `href` в этом файле XML. Вызов `document()` возвращает набор узлов, представляющий заданный документ. Каждый атрибут `href` ссылается на документ с информацией о заказе, который выглядит примерно так:

```
<?xml version="1.0" ?>
<!-- po38293.xml -->
<purchase-order id="38293">
  <date year="2001" month="9" day="8"/>
  <customer id="4738" level="Basic">
    <address type="business">
      <name>
        <title>Ms.</title>
        <first-name>Amanda</first-name>
        <last-name>Reckonwith</last-name>
      </name>
      <street>930-A Chestnut Street</street>
      <city>Lynn</city>
      <state>MA</state>
      <zip>02930</zip>
    </address>
    <address type="ship-to"/>
  </customer>
  <items>
    <item part-no="23813-03-CDK">
      <name>Cucumber Decorating Kit</name>
      <qty>1</qty>
      <price>29.95</price>
    </item>
  </items>
</purchase-order>
```

Наша таблица стилей должна находить все заказы, в которых клиент заказал более одного товара. Вот как она выглядит:

```
<?xml version="1.0"?>
<!-- document.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the document() function:&#xA;</xsl:text>

    <xsl:for-each select="/collection/doc">
      <xsl:variable name="latestDoc" select="document(@href)"/>
      <xsl:if
        test="count($latestDoc/purchase-order/items/item) &gt; 1">
        <xsl:text>&#xA;  </xsl:text>
        <xsl:value-of
          select="$latestDoc/purchase-order/customer/
                  address/name/first-name"/>
        <xsl:text> </xsl:text>
        <xsl:value-of
          select="$latestDoc/purchase-order/customer/
                  address/name/last-name"/>
        <xsl:text>&#xA;    ordered </xsl:text>
```

```
        <xsl:value-of
          select="count($latestDoc/purchase-order/items/item)"/>
        <xsl:text> items on </xsl:text>
        <xsl:value-of
          select="$latestDoc/purchase-order/date/@month"/>
        <xsl:text>/</xsl:text>
        <xsl:value-of
          select="$latestDoc/purchase-order/date/@day"/>
        <xsl:text>/</xsl:text>
        <xsl:value-of
          select="$latestDoc/purchase-order/date/@year"/>
        <xsl:text> - see P.O. #</xsl:text>
        <xsl:value-of
          select="$latestDoc/purchase-order/@id"/>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

При выполнении таблицы стилей будет получен следующий результат:

```
A test of the document() function:

  Chester Hasbrouck Frisby
    ordered 3 items on 6/19/2001 - see P.O. #38292
  Natalie Attired
    ordered 2 items on 4/21/2001 - see P.O. #38294
```

В этой таблице стилей функция `document()` используется для открытия всех заказов, упоминаемых в *polist.xml*. Для всех заказов, в которых клиент заказал более одного товара (`count($latestDoc/purchase-order/items/item) > 1`), выводятся некоторые сведения из заказа. Обратите внимание на повторную инициализацию переменной `$latestDoc` при переборе всех атрибутов `href`; если мы обнаружим документ, нуждающийся в обработке, сохранение дерева в переменной позволит избежать многократных вызовов `document()` для получения нужной информации.

[2.0] document-uri()

Для заданного узла возвращает URI-адрес документа.

Синтаксис

```
xs:anyURI? document-uri(node()?)
```

Ввод

Узел.

Вывод

Если узел не является узлом документа, `document-uri()` возвращает пустую последовательность. В противном случае функция возвращает свойство URI-адреса документа для заданного узла. Если аргумент является пустой последовательностью, `document-uri()` возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 2 «Accessors».

Пример

Короткая таблица стилей, возвращающая URI-адрес обрабатываемого документа XML:

```
<?xml version="1.0"?>
<!-- document-uri.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the document-uri() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  The document URI for the </xsl:text>
    <xsl:text>root element is:&#xA;    </xsl:text>
    <xsl:value-of select="document-uri(/)"/>
  </xsl:template>

</xsl:stylesheet>
```

При выполнении таблицы стилей для файла *blank.xml* будет получен следующий результат:

```
A test of the document-uri() function:

  The document URI for the root element is:
    file:/C:/projects/XSLTbookV2/AppendixC/blank.xml
```

element-available()

Проверяет доступность элемента для процессора XSLT. Функция позволяет строить таблицы стилей, корректно реагирующие на недоступность некоторого расширяющего элемента для документа XML.

Синтаксис

```
[1.0] boolean element-available($elementName as string)
[2.0] xs:boolean element-available($elementName as xs:string)
```

Ввод

Имя элемента. Имя должно уточняться пространством имен; если URI-адрес пространства имен совпадает с URI-адресом пространства имен XSLT, значит, имя элемента относится к элементу, определяемому в XSLT. В противном случае имя относится к элементу расширения. Если имя элемента ассоциировано с пустым URI-адресом пространства имен, функция `element-available()` возвращает `false`. *[2.0]* Если аргумент невозможно преобразовать в действительное полное имя (QName), процессор XSLT выдает ошибку.

Вывод

Логическое значение `true`, если элемент доступен; `false` в противном случае.

Где определяется

[1.0] XSLT, раздел 15 «Fallback».

[2.0] XSLT, раздел 18.2 «Extension Instructions».

Пример

Для демонстрации работы функции `element-available()` будет использоваться следующий документ XML:

```
<?xml version="1.0"?>
<!-- chapterlist.xml -->
<book>
  <title>XSLT</title>
  <chapter>
    <title>Getting Started</title>
    <para>If this chapter had any text, it would appear here.</para>
  </chapter>
  <chapter>
    <title>The Hello World Example</title>
    <para>If this chapter had any text, it would appear here.</para>
  </chapter>
  ...
  <chapter>
    <title>Combining XML Documents</title>
    <para>If this chapter had any text, it would appear here.</para>
  </chapter>
</book>
```

Наша таблица стилей пытается использовать расширение Xalan-J `<redirect:write>` для записи элементов `<chapter>` в разные файлы. Если элемент недоступен, вся информация записывается в один файл HTML. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- element-available.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:redirect="org.apache.xalan.xslt.extensions.Redirect"
  extension-element-prefixes="redirect">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <xsl:choose>
      <xsl:when test="element-available('redirect:write')">
        <xsl:for-each select="/book/chapter">
          <redirect:write select="concat('chapter', position(), '.html')">
            <html>
              <head>
                <title><xsl:value-of select="title"/></title>
              </head>
              <body>
                <h1><xsl:value-of select="title"/></h1>
                <xsl:apply-templates select="para"/>
                <xsl:if test="not(position()=1)">
                  <p>
                    <a href="chapter{position()-1}.html">Previous</a>
                  </p>
```

```
                </xsl:if>
                <xsl:if test="not(position()=last())">
                  <p>
                    <a href="chapter{position()+1}.html">Next</a>
                  </p>
                </xsl:if>
              </body>
            </html>
          </redirect:write>
        </xsl:for-each>
      </xsl:when>
      <xsl:otherwise>
        <html>
          <head>
            <title><xsl:value-of select="/book/title"/></title>
          </head>
          <xsl:for-each select="/book/chapter">
            <h1><xsl:value-of select="title"/></h1>
            <xsl:apply-templates select="para"/>
          </xsl:for-each>
        </html>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template match="para">
    <p><xsl:apply-templates select="*|text()"/></p>
  </xsl:template>

</xsl:stylesheet>
```

Если элемент `<redirect:write>` **доступен, для каждого элемента** `<chapter>` **создается отдельный файл. Файлы выглядят так:**

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
      <title>XPath</title>
   </head>
   <body>
      <h1>XPath</h1>
      <p>If this chapter had any text, it would appear here.</p>
      <p><a href="chapter2.html">Previous</a></p>
      <p><a href="chapter4.html">Next</a></p>
   </body>
</html>
```

При отображении в броузере файл выглядит так, как показано на рис. C.1.

Щелчок на ссылке `Previous` **открывает файл** `chapter2.html`**, а ссылка** `Next` **ведет к файлу** `chapter4.html`**.**

Несмотря на небольшие различия в форматах сообщений, содержимое разных файлов HTML практически полностью совпадает.

При использовании процессора Saxon, не поддерживающего элемент `<redirect: write>`**, мы получаем единый файл HTML следующего вида:**

Рис. C.1. *Пример выходного файла в формате HTML*

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>XSLT</title>
   </head>
   <h1>Getting Started</h1>
   <p>If this chapter had any text, it would appear here.</p>
   ...
   <h1>Combining XML Documents</h1>
   <p>If this chapter had any text, it would appear here.</p>
</html>
```

На рис. C.2 показано, как этот файл выглядит в броузере.

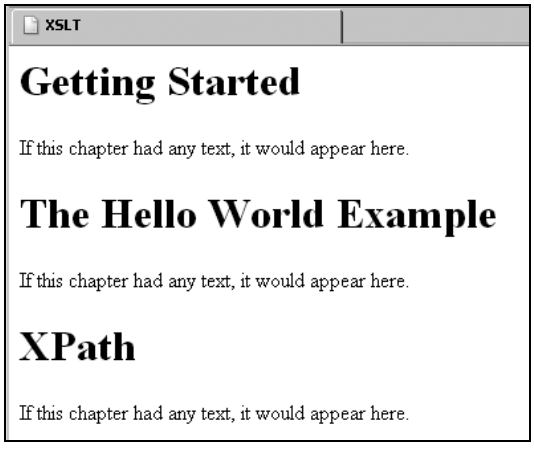


Рис. C.2. *Документ HTML со списком всех глав*

MSXSL и ядро AltovaXML выдают те же результаты.

В этом примере функция `element-available()` позволяет проверить доступные возможности процессора XSLT и корректно отреагировать на их изменение.

[2.0] empty()

Возвращает `true`, если аргумент является пустой последовательностью, и `false` в противном случае.

Синтаксис

```
xs:boolean empty(item()*)
```

Ввод

Последовательность объектов.

Вывод

`true`, если аргумент является пустой последовательностью; `false` в противном случае. Вызовы `empty($sequence)` и `count($sequence) = 0` логически эквивалентны, однако вызов `empty()`, вероятно, будет более эффективным.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.1 «General Functions and Operators on Sequences».

Пример

Таблица стилей для демонстрации работы функции `empty()`:

```
<?xml version="1.0"?>
<!-- empty.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <xsl:variable name="emptySequence" as="item()*">
      <xsl:sequence select="()"/>
    </xsl:variable>

    <xsl:variable name="singleton" as="item()*">
      <xsl:sequence select="(3)"/>
    </xsl:variable>

    <xsl:variable name="longSequence" as="item()*">
      <xsl:sequence
        select="(3, 4, 5, current-date(), current-time(), 8, 'blue',
                 'red', xs:float(3.14), 42, xs:date('1995-04-21'))"/>
    </xsl:variable>

    <xsl:text>&#xA;Here are some tests of the empty() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  Our first sequence is:&#xA;&#xA;    </xsl:text>
    <xsl:value-of select="$emptySequence" separator="&#xA;    "/>
    <xsl:text>&#xA;&#xA;</xsl:text>
    <xsl:value-of select="if (empty($emptySequence))
                          then 'This is the empty sequence!'
```

```
                        else 'This is not the empty sequence!'"/>
    <xsl:text>&#xA;&#xA;</xsl:text>

    <xsl:text>&#xA;  Our next sequence is:&#xA;&#xA;    </xsl:text>
    <xsl:value-of select="$singleton" separator="&#xA;    "/>
    <xsl:text>&#xA;&#xA;</xsl:text>
    <xsl:value-of select="if (empty($singleton))
                        then 'This is the empty sequence!'
                        else 'This is not the empty sequence!'"/>
    <xsl:text>&#xA;&#xA;</xsl:text>

    <xsl:text>&#xA;  Our final sequence is:&#xA;&#xA;    </xsl:text>
    <xsl:value-of select="$longSequence" separator="&#xA;    "/>
    <xsl:text>&#xA;&#xA;</xsl:text>
    <xsl:value-of select="if (empty($longSequence))
                        then 'This is the empty sequence!'
                        else 'This is not the empty sequence!'"/>
    <xsl:text>&#xA;&#xA;</xsl:text>

  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Here are some tests of the empty() function:

  Our first sequence is:

This is the empty sequence!

  Our next sequence is:
    3

This is not the empty sequence!

  Our final sequence is:
    3
    4
    5
    2008-03-04-05:00
    11:04:14.25-05:00
    8
    blue
    red
    3.14
    42
    1995-04-21

This is not the empty sequence!
```

[2.0] encode-for-uri()

Кодирует заданную строку для использования в URI-адресе.

Синтаксис

```
xs:string encode-for-uri($uri-part as xs:string?)
```

Ввод

Строка, содержащая кодируемое значение.

Вывод

Входная строка, в которой соответствующие символы кодируются с префиксом %. Кодируются все символы, *за исключением* следующих: букв A–Z в верхнем и нижнем регистрах, цифр 0–9, дефиса («–» или -), символа подчеркивания («_» или _), точки («.» или &#2E;) и тильды («~» или ~). Например, пробел (&#x20) кодируется в виде %20. Все шестнадцатеричные цифры записываются в верхнем регистре, так что символ á (á) кодируется в виде %E1.

Помните, что функция encode-for-uri() кодирует все символы, поэтому ее следует вызывать только для тех частей URI-адреса, для которых это действительно необходимо. Например, вызов encode-for-uri('http://www.oreilly.com/') возвращает http%3A%2F%2Fwww.oreilly.com%2F, а это почти наверняка *не* то, что вы хотели получить.

Если значение аргумента является пустой последовательностью, функция возвращает строку нулевой длины.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.4 «Functions on String Values».

Пример

Следующая таблица стилей демонстрирует два примера, позаимствованные из спецификации XQuery 1.0 and XPath 2.0 Functions and Operators:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- encode-for-uri.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the encode-for-uri() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  encode-for-uri</xsl:text>
    <xsl:text>('http://www.oreilly.com/') = &#xA;    </xsl:text>
    <xsl:value-of
      select="encode-for-uri('http://www.oreilly.com/')"/>

    <xsl:text>&#xA;&#xA;  concat('http://www.example.com/', </xsl:text>
    <xsl:text>&#xA;          encode-for-uri</xsl:text>
    <xsl:text>('~b&#xE9;b&#xE9;')) = &#xA;    </xsl:text>
    <xsl:value-of
      select="concat('http://www.example.com/',
              encode-for-uri('~b&#xE9;b&#xE9;'))"/>
    <xsl:text>&#xA;&#xA;  concat('http://www.example.com/', </xsl:text>
    <xsl:text>&#xA;          encode-for-uri</xsl:text>
    <xsl:text>('100% organic')) = &#xA;    </xsl:text>
    <xsl:value-of
      select="concat('http://www.example.com/',
              encode-for-uri('100% organic'))"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the encode-for-uri() function:

  encode-for-uri('http://www.oreilly.com/') =
    http%3A%2F%2Fwww.oreilly.com%2F

  concat('http://www.example.com/',
         encode-for-uri('~bйbй')) =
    http://www.example.com/~b%C3%A9b%C3%A9

  concat('http://www.example.com/',
         encode-for-uri('100% organic')) =
    http://www.example.com/100%25%20organic
```

См. также

Описания функций *[2.0]* `escape-html-uri()` и *[2.0]* `iri-to-uri()`.

[2.0] ends-with()

Получает две строки и необязательную схему упорядочения; возвращает `true`, если первая строка завершается символами второй строки.

Синтаксис

```
xs:boolean ends-with(xs:string?, xs:string?)
xs:boolean ends-with(xs:string?, xs:string?, $collation as xs:string)
```

Ввод

Две строки и необязательное имя алгоритма упорядочения. Алгоритм упорядочения определяет, каким образом осуществляется сравнение символов. Позаимствуем пример из спецификации: алгоритм упорядочения может определять, что символы `ss` и немецкий символ ß («длинное s») равны.

Вывод

Если оба аргумента не являются строками нулевой длины, а первая строка завершается второй строкой, `end-with()` возвращает логическое значение `true`; в остальных случаях возвращается `false`. Если вторая строка имеет нулевую длину, `ends-with()` возвращает `true`. Если обе строки, первая и вторая, имеют нулевую длину, `ends-with()` возвращает `true`. Если первая строка является строкой нулевой длины, а вторая — нет, `ends-with()` возвращает `false`.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.5 «Functions Based on Substring Matching».

Пример

Короткая таблица стилей для демонстрации работы функции `ends-with()`:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- ends-with1.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="string1" select="'Have a nice day'"/>
    <xsl:variable name="string2" select="'Have a nice day!'"/>
    <xsl:variable name="string3" select="'day'"/>

    <xsl:text>&#xA;A test of the ends-with() function:&#xA;</xsl:text>

    <xsl:text>  ends-with('</xsl:text>
    <xsl:value-of select="$string1"/>
    <xsl:text>', '</xsl:text>
    <xsl:value-of select="$string3"/>
    <xsl:text>') = </xsl:text>
    <xsl:value-of select="ends-with($string1, $string3)"/>
    <xsl:text>&#xA;</xsl:text>

    <xsl:text>  ends-with('</xsl:text>
    <xsl:value-of select="$string2"/>
    <xsl:text>', '</xsl:text>
    <xsl:value-of select="$string3"/>
    <xsl:text>') = </xsl:text>
    <xsl:value-of select="ends-with($string2, $string3)"/>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
A test of the ends-with() function:
  ends-with('Have a nice day', 'day') = true
  ends-with('Have a nice day!', 'day') = false
```

Функция `ends-with` **входит в ограниченный набор функций, которым может передаваться необязательный аргумент со схемой упорядочения. Способ определения схемы упорядочения зависит от процессора XSLT. В Saxon для этого используется URI-адрес, включающий имя класса Java, выполняющего упорядочение. В следующем примере сравниваются немецкие слова** `Straße` **и** `Strasse`**. Со схемой упорядочения по умолчанию эти два слова различаются; со схемой упорядочения для немецкого языка они совпадают. Таблица стилей выглядит так:**

```
<?xml version="1.0" encoding="utf-8"?>
<!-- ends-with2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="string1" select="'Stra&#xDF;e'"/>
    <xsl:variable name="string2" select="'Strasse'"/>
    <xsl:text>&#xA;A test of the ends-with() function:&#xA;</xsl:text>

    <xsl:text>  ends-with('</xsl:text>
    <xsl:value-of select="$string1"/>
    <xsl:text>', '</xsl:text>
    <xsl:value-of select="$string2"/>
    <xsl:text>') = </xsl:text>
    <xsl:value-of select="ends-with($string1, $string2)"/>
```

```
    <xsl:text>&#xA;</xsl:text>

    <xsl:text>  ends-with('</xsl:text>
    <xsl:value-of select="$string1"/>
    <xsl:text>', '</xsl:text>
    <xsl:value-of select="$string2"/>
    <xsl:text>', [German collation]) = </xsl:text>
    <xsl:value-of
      select="ends-with($string1, $string2,
              concat('http://saxon.sf.net/collation?',
                     'class=com.oreilly.xslt.GermanCollation;'))"/>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
A test of the ends-with() function:
  ends-with('Straße', 'Strasse') = false
  ends-with('Straße', 'Strasse', [German collation]) = true
```

С определенной нами пользовательской схемой упорядочения два варианта написания `Strasse` считаются одинаковыми; со схемой упорядочения по умолчанию они различаются. (Чтобы листинг поместился на странице, мы объединили две половины URI-адреса схемы упорядочения Saxon при помощи функции `concat()`.) За дополнительной информацией обращайтесь к разделу «Функция document() и сортировка» главы 8.

[2.0] error()

Инициирует ошибку; эквивалент инициирования исключений в Java, C++, Ruby и других языках.

Синтаксис

```
none error()
none error($error as xs:QName)
none error($error as xs:QName?, $description as xs:string)
none error($error as xs:QName?, $description as xs:string,
           $error-object as item()*)
```

Ввод

Функция имеет четыре разные сигнатуры. Три необязательных параметра, определяемых в разделе «Синтаксис», представляют полное имя, связанное с ошибкой, значение `xs:string` и последовательность объектов. Способ обработки ввода определяется реализацией, так что полезные значения этих аргументов зависят от конкретного процессора XSLT.

Вывод

Функция вообще не возвращает управления; она передает ошибку внешней среде для обработки. Способ передачи внешней среде полного имени, строки или последовательности, полученных функцией, зависит от реализации.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 3 «The Error Function». В приложении C к этой спецификации определяется ряд условий и кодов ошибок. Например, `err:FODT0003` – код ошибки недействительного часового пояса. Пространство имен `err` связывается с URI-адресом `http://www.w3.org/2005/xqt-errors` – мы используем его в следующих примерах. При желании вы можете определять собственные сообщения и коды ошибок.

Пример

Мы рассмотрим четыре таблицы стилей, инициирующие ошибки – по одной для каждой из четырех сигнатур функции `error()`. В первой таблице стилей используется простейший вызов `error()`:

```
<?xml version="1.0"?>
<!-- error1.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Test #1 of the error() function:</xsl:text>

    <xsl:value-of select="error()"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
  FOER0000: Error signalled by application call on error()
Transformation failed: Run-time errors were reported
```

Код ошибки `FOER0000` определяется в спецификации XQuery 1.0 and XPath 2.0 Functions and Operators как «неопознанная ошибка».

Во втором примере используется полное имя, ссылающееся на конкретный код ошибки из спецификации:

```
<?xml version="1.0"?>
<!-- error2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Test #2 of the error() function:</xsl:text>

    <xsl:value-of
      select="error(QName('http://www.w3.org/2005/xqt-errors',
              'err:FORX0004'))"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
  FORX0004: Error signalled by application call on error()
Transformation failed: Run-time errors were reported
```

На этот раз в результате присутствует конкретный код ошибки FORX0004. Мы перейдем к третьему примеру, в котором при вызове error() передается строка с кратким описанием ошибки:

```
<?xml version="1.0"?>
<!-- error3.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Test #3 of the error() function:</xsl:text>

    <xsl:value-of
      select="error(QName('http://www.w3.org/2005/xqt-errors',
              'err:FORX0004'),
              'Invalid replacement string.')"/>
  </xsl:template>

</xsl:stylesheet>
```

На этот раз результат более содержателен; в сообщение об ошибке включается переданная нами строка:

```
  FORX0004: Invalid replacement string.
Transformation failed: Run-time errors were reported
```

В спецификации код ошибки FORX0004 определяется как признак «недействительная строка замены», поэтому мы помещаем соответствующее сообщение в строку ошибки. В Saxon переданная функции error() строка включается в вывод; это позволяет при необходимости вывести любую информацию на наше усмотрение.

В последнем примере в выходные данные включается значение xs:dateTime:

```
<?xml version="1.0"?>
<!-- error4.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Test #4 of the error() function:</xsl:text>

    <xsl:value-of
      select="error(QName('http://www.w3.org/2005/xqt-errors',
              'err:FORX0004'),
              'Invalid replacement string.',
              current-dateTime())"/>
  </xsl:template>

</xsl:stylesheet>
```

Однако результатат ничем не отличается от предыдущего:

```
  FORX0004: Invalid replacement string.
Transformation failed: Run-time errors were reported
```

Заглянув в документацию Saxon, мы обнаруживаем, что последовательность объектов, передаваемая функции error(), предназначается для нетривиального использования, не ограничивающегося простым выводом сообщений на консоль. Например,

эти данные доступны для функции saxon:try(), поддерживаемой схемосовместимой версией Saxon. И снова практически все подробности работы функции error() определяются реализацией; за дополнительной информацией обращайтесь к документации своего процессора.

[2.0] escape-html-uri()

Для заданного URI-адреса для HTML возвращает тот же URI-адрес, в котором все символы, не входящие в набор ASCII, кодируются как символы UTF-8.

Синтаксис

```
xs:string escape-html-uri(xs:string?)
```

Ввод

Значение xs:string, представляющее URI-адрес для HTML.

Вывод

Входная строка, в которой все символы за пределами диапазона от   до ~ кодируются как символы UTF-8. При преобразовании в UTF-8 символ кодируется октетом в форме %HH, где HH – шестнадцатеричное представление октета. Функция escape-html-uri() генерирует шестнадцатеричные символы в верхнем регистре. Функция соответствует правилам преобразования символов, не входящим в набор ASCII, определенным в спецификации HTML 4.0.

Если аргумент является пустой последовательностью, функция возвращает строку нулевой длины.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.4 «Functions on String Values».

Пример

В этой таблице стилей функция escape-html-uri() используется для правильного кодирования символов в URI-адресе (в данном случае символа é) в UTF-8. (Пример позаимствован из спецификации XQuery 1.0 and XPath 2.0 Functions and Operators.)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- escape-html-uri.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the escape-html-uri() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  escape-html-uri</xsl:text>
    <xsl:text>('http://www.example.com/~bйbй') = &#xA;    </xsl:text>
    <xsl:value-of
      select="escape-html-uri('http://www.example.com/~bйbй')"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the escape-html-uri() function:

  escape-html-uri('http://www.example.com/~bйbй') =
    http://www.example.com/~b%C3%A9b%C3%A9
```

Обратите внимание: однобайтовый символ в исходном URL-адресе (é) в выходных данных заменяется двухбайтовым символом UTF-8 (%C3%A9).

См. также

Описания функций *[2.0]* encode-for-uri() и *[2.0]* iri-to-uri().

[2.0] exactly-one()

Выдает ошибку, если аргумент не является синглетом (последовательностью, содержащей ровно один объект). Учтите, что exactly-one() завершает обработку таблицы стилей. Если вам потребуется более гибкий подход, используйте функцию count() для определения количества объектов в последовательности.

Синтаксис

```
item()+ exactly-one(item()*)
```

Ввод

Последовательность.

Вывод

Если входная последовательность содержит ровно один объект, функция возвращает входную последовательность. В противном случае exactly-one() выдает ошибку.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.2 «Functions That Test the Cardinality of Sequences». Дополнительная информация об этой функции приводится в XQuery 1.0 and XPath 2.0 Formal Semantics, раздел 7.2 «Standard Functions with Specific Static Typing Rules».

Пример

Функция exactly-one() может оказаться полезной для работы со статической типизацией XSLT 2.0. Любая последовательность либо пуста, либо является синглетной, либо содержит несколько объектов. Если вам потребуется проверить кардинальное число (количество объектов) в последовательности во время выполнения, функция exactly-one() поможет выдержать требование о синглетности последовательности. Однако если аргумент не содержит ровно один объект, exactly-one() завершает обработку таблицы стилей; другое, намного более гибкое решение основано на предварительной проверке количества объектов последовательности функцией count().

Короткая таблица стилей, в которой функция exactly-one() вызывается с синглетной последовательностью:

```
<?xml version="1.0"?>
<!-- exactly-one.xsl -->
<xsl:stylesheet version="2.0"
```

```
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:output method="text"/>

    <xsl:template match="/">

      <xsl:text>&#xA;Here is a test of the exactly-one() </xsl:text>
      <xsl:text>function:&#xA;</xsl:text>

      <xsl:variable name="singleton" as="item()*">
        <xsl:sequence select="0"/>
      </xsl:variable>

      <xsl:text>&#xA;  Our sequence is:&#xA;&#xA;    </xsl:text>
      <xsl:value-of select="$singleton" separator="&#xA;    "/>

      <xsl:if test="count(exactly-one($singleton))">
        <xsl:text>&#xA;&#xA;  Our test sequence has exactly </xsl:text>
        <xsl:text>one item!&#xA;</xsl:text>
      </xsl:if>
    </xsl:template>

</xsl:stylesheet>
```

В атрибуте `test` **элемента** `<xsl:if>` **по результатам вызова** `exactly-one()` **необходимо сгенерировать логическое значение. Простой (но неправильный) способ:**

```
<xsl:if test="exactly-one($singleton)">  <!-- doesn't work! -->
```

Если функция `exactly-one()` **не инициирует ошибку, она возвращает последовательность, содержащую ровно один объект. Казалось бы, последовательность из одного объекта всегда будет интерпретироваться как** `true`**, но это не так. В приведенном примере синглетная последовательность (**`0`**) интерпретируется как** `false`**, потому что атомарное значение** `0` **интерпретируется как** `false`**. Использование функции** `count()` **обеспечивает получение осмысленного результата. (Для синглетной последовательности** `count()` **всегда возвращает** `1`**, что всегда соответствует** `true`**.)**

Следует также помнить, что многие типы данных не преобразуются в логические значения. При попытке использования синглетной последовательности `(xs:date('1995-04-21'))` **процессор XSLT инициирует ошибку, потому что значения** `xs:date` **не преобразуются в логические.**

Результат выглядит так:

```
Here is a test of the exactly-one() function:

  Our sequence is:

    0

  Our test sequence has exactly one item!
```

См. также

Описания функций `count()`, ***[2.0]*** `empty()`, ***[2.0]*** `one-or-more()` **и** ***[2.0]*** `zero-or-one()`, **описание оператора XPath 2.0** `treat as` **(раздел «[2.0] Операторы типов данных – instance of, castable as, cast as и treat as» главы 3).**

[2.0] exists()

Возвращает `true`, если входная последовательность не пуста; `false` в противном случае.

Синтаксис

```
xs:boolean exists(item()*)
```

Ввод

Последовательность.

Вывод

Если последовательность не пуста, `exists()` возвращает `true`; в противном случае возвращается `false`.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.1 «General Functions and Operators on Sequences».

Пример

В следующей таблице стилей функция `exists()` вызывается для трех последовательностей. Первая последовательность пуста, вторая является синглетной, а третья содержит несколько объектов:

```
<?xml version="1.0"?>
<!-- exists.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <xsl:text>&#xA;Here is a test of the exists() </xsl:text>
    <xsl:text>function:&#xA;&#xA;  </xsl:text>

    <xsl:variable name="emptySequence" as="item()*">
      <xsl:sequence select="()"/>
    </xsl:variable>

    <xsl:variable name="singleton" as="item()*">
      <xsl:sequence select="(3)"/>
    </xsl:variable>

    <xsl:variable name="longSequence" as="item()*">
      <xsl:sequence
        select="(3, 4, 5, current-date(), current-time(), 8, 'blue',
                 'red', xs:float(3.14), 42, xs:date('1995-04-21'))"/>
    </xsl:variable>

    <xsl:choose>
      <xsl:when test="exists($emptySequence)">
        <xsl:text>The empty sequence does exist!</xsl:text>
      </xsl:when>
      <xsl:otherwise>
```

```
        <xsl:text>The empty sequence doesn't exist!</xsl:text>
      </xsl:otherwise>
    </xsl:choose>

    <xsl:text>&#xA;  </xsl:text>
    <xsl:value-of select="if (exists($singleton))
                          then 'The singleton sequence does exist!'
                          else 'The singleton sequence doesn''t exist!'"/>
    <xsl:text>&#xA;  </xsl:text>
    <xsl:value-of select="if (exists($longSequence))
                          then 'The long sequence does exist!'
                          else 'The long sequence doesn''t exist!'"/>
  </xsl:template>

</xsl:stylesheet>
```

Первая проверка `exists()` использует для принятия решений элемент XSLT 1.0 `<choose>`, а в остальных случаях функция `exists()` применяется в конструкции XPath 2.0 `if`. Таблица стилей выдает следующий результат:

```
Here is a test of the exists() function:

  The empty sequence doesn't exist!
  The singleton sequence does exist!
  The long sequence does exist!
```

false()

Всегда возвращает логическое значение `false`. Вспомните, что строки «true» и «false» не имеют особой интерпретации в XSLT; любая строка, длина которой больше нуля, считается истинной. Эта функция (вместе с функцией `true()`) позволяет генерировать логические значения там, где потребуется.

Синтаксис

```
[1.0] boolean false()
[2.0] xs:boolean false()
```

Ввод

Нет.

Вывод

Логическое значение `false`.

Где определяется

XPath, раздел 4.3 «Boolean Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 9.1 «Additional Boolean Constructor Functions».

Пример

Короткий пример использования функции `false()`:

```
<?xml version="1.0"?>
<!-- false.xsl -->
```

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the false() function:&#xA;&#xA;</xsl:text>

    <xsl:text>  false() returned </xsl:text>
    <xsl:value-of select="false()"/>
    <xsl:text>!</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

Для любого документа XML эта таблица стилей выдает вполне предсказуемый результат:

```
A test of the false() function:

  false() returned false!
```

floor()

Возвращает наибольшее целое число, не превышающее аргумента.

Синтаксис

```
[1.0] number floor(number)
[2.0] numeric? floor(numeric?)
```

Ввод

Число.

[1.0] Если аргумент не является числом, он преобразуется в число так, как при обработке функцией `number()`. Если аргумент не может быть преобразован в число, функция `floor()` возвращает значение `NaN` («нечисло»).

[2.0] В XSLT 2.0 аргумент должен относиться к одному из четырех числовых типов (`xs:float`, `xs:decimal`, `xs:double` или `xs:integer`). При нарушении этого условия процессор XSLT выдает ошибку. Результат вызова функции `floor()` относится к тому же типу, что и аргумент.

В XSLT 2.0 аргумент должен быть числом. Для любых других значений процессор XSLT выдает ошибку.

Вывод

Наибольшее целое, не превышающее аргумента.

[1.0] В XSLT 1.0 функция `floor()` возвращает `NaN`, если аргумент не может быть преобразован в число.

[2.0] В XSLT 2.0 функция `floor()` выдает ошибку, если аргумент не может быть преобразован в число.

Где определяется

[1.0] XPath, раздел 4.4 «Number Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 6.4 «Functions on Numeric Values».

Пример

Следующая таблица стилей демонстрирует вызов функции floor() для различных значений:

```
<?xml version="1.0"?>
<!-- floor.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the floor() function:&#xA;&#xA;</xsl:text>

    <xsl:text>  floor(7.983) = </xsl:text>
    <xsl:value-of select="floor(7.983)"/>

    <xsl:text>&#xA;  floor(-7.893) = </xsl:text>
    <xsl:value-of select="floor(-7.893)"/>

    <xsl:text>&#xA;  floor('blue') = </xsl:text>
    <xsl:value-of version="1.0" select="floor('blue')"/>

  </xsl:template>

</xsl:stylesheet>
```

При преобразовании документа XML с этой таблицей стилей будет получен следующий результат:

```
Tests of the floor() function:

  floor(7.983) = 7
  floor(-7.893) = -8
  floor('blue') = NaN
```

В последнем тесте функции floor() в элементе <xsl:value-of> указывается атрибут version="1.0". В режиме XSLT 1.0 возвращается результат NaN («нечисло»). При обработке элемента <xsl:value-of> в режиме XSLT 2.0 таблица стилей вообще не работает. Если вся таблица стилей переводится на XSLT версии 1.0, вам придется удалить атрибут version из элемента <xsl:value-of>.

[2.0] format-date()

Форматирует значение xs:date в соответствии с форматной строкой.

Синтаксис

```
xs:string? format-date(xs:date?, xs:string)
xs:string? format-date(xs:date?, xs:string, $language as xs:string?,
                       $calendar as xs:string?,
                       $country as xs:string?)
```

Ввод

В первой разновидности эта функция получает необязательное значение xs:date и форматную строку. Если значение xs:date не указано, используется текущая дата. Второй разновидности передаются строки, представляющие желательные коды языка, календаря и страны для выходного значения.

Далее перечислены коды, определенные для форматных строк. Значения по умолчанию определяют интерпретацию кодов процессором без дополнительных модификаторов. Иначе говоря, `[Y]` – то же самое, что `[Y1]`.

`Y` Год. По умолчанию используется значение `[Y1]`, генерирующее год из четырех цифр.

`M` Месяц. По умолчанию используется значение `[M1]`, генерирующее номер месяца из одной или двух цифр.

`D` День месяца. По умолчанию используется значение `[D1]`, генерирующее номер дня из одной или двух цифр.

`d` День года. По умолчанию используется значение `[d1]`, генерирующее номер дня из одной, двух или трех цифр (например, для 26 июля невисокосного года генерируется значение `207`).

`F` День недели. По умолчанию используется значение `[Fn]`, генерирующее название дня недели в нижнем регистре (например, `wednesday`).

`W` Неделя года. По умолчанию используется значение `[W1]`, генерирующее номер недели в году из одной или двух цифр.

`w` Неделя месяца. По умолчанию используется значение `[w1]`, генерирующее номер недели в месяце из одной цифры.

`Z` Часовой пояс, заданный смещением от всемирного координированного времени UTC. По умолчанию используется значение `[Z1]`, генерирующее смещение от UTC в часах и минутах (например, `-04:00`).

`z` Часовой пояс, заданный смещением от гринвичского среднего времени GMT. По умолчанию используется значение `[z1]`, генерирующее смещение от GMT (например, `GMT-4`).

За полной информацией о форматных строках даты, времени и чисел обращайтесь к приложению F.

Вывод

Строка, представляющая значение `xs:date`, отформатированное в соответствии с форматной строкой. Если аргумент `xs:date` является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XSLT, раздел 16.5 «Formatting Dates and Times».

Пример

Следующая таблица стилей демонстрирует различные способы форматирования значения `xs:date` и его компонентов:

```
<?xml version="1.0"?>
<!-- format-date.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the format-date() function:&#xA;</xsl:text>
```

```
    <xsl:variable name="today" select="current-date()"/>
    <xsl:text>&#xA;  The current date is: </xsl:text>
    <xsl:value-of select="$today"/>
    <xsl:variable name="longFormat"
      select="concat('Today is [FNn], [MNn] [D1], [Y], &#xA;      ',
              'the [D1o] day of the [W1o] week of the year,',
              '&#xA;      the [w1o] week of the month,',
              '&#xA;      in the [zWw] time zone.')"/>
    <xsl:text>&#xA;&#xA;    The year - format-date</xsl:text>
    <xsl:text>($today, '[Y]'): </xsl:text>
    <xsl:value-of select="format-date($today, '[Y1111]')"/>
    <xsl:text>&#xA;&#xA;    The two-digit year - </xsl:text>
    <xsl:text>format-date($today, '[Y11]'): </xsl:text>
    <xsl:value-of select="format-date($today, '[Y11]')"/>
    <xsl:text>&#xA;&#xA;    The month - format-date(</xsl:text>
    <xsl:text>$today, '[M11]'): </xsl:text>
    <xsl:value-of select="format-date($today, '[M11]')"/>
    <xsl:text>&#xA;&#xA;    The month in German words - </xsl:text>
    <xsl:text>format-date($today, '[MWw]', 'de', (), ()): </xsl:text>
    <xsl:text>&#xA;      </xsl:text>
    <xsl:value-of
      select="format-date($today, '[MWw]', 'de', (), ())"/>
    <xsl:text>&#xA;&#xA;    The day in words - format-date(</xsl:text>
    <xsl:text>$today, '[DWw]'): &#xA;      </xsl:text>
    <xsl:value-of select="format-date($today, '[DWw]')"/>
    <xsl:text>&#xA;&#xA;    The day in German words - </xsl:text>
    <xsl:text>format-date($today, '[Dw]', 'de', (), ()): </xsl:text>
    <xsl:text>&#xA;      </xsl:text>
    <xsl:value-of select="format-date($today, '[Dw]', 'de', (), ())"/>
    <xsl:text>&#xA;&#xA;    Using format-date($today, '[dwo]'):</xsl:text>
    <xsl:text>&#xA;      It's the </xsl:text>
    <xsl:value-of select="format-date($today, '[dwo]')"/>
    <xsl:text> day of the year.</xsl:text>
    <xsl:text>&#xA;&#xA;    Using format-date($today, '[d1o]'):</xsl:text>
    <xsl:text>&#xA;      It's the </xsl:text>
    <xsl:value-of select="format-date($today, '[d1o]')"/>
    <xsl:text> day of the year.</xsl:text>

    <xsl:text>&#xA;&#xA;  The grand finale:</xsl:text>
    <xsl:text>&#xA;    </xsl:text>
    <xsl:value-of select="format-date($today, $longFormat)"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат выглядит так:

```
Tests of the format-date() function:

  The current date is: 2006-07-26-04:00

    The year - format-date($today, '[Y]'): 2006

    The two-digit year - format-date($today, '[Y11]'): 06

    The month - format-date($today, '[M11]'): 07

    The month in German words - format-date($today, '[MWw]', 'de', (), ()):
      Sieben
```

```
    The day in words - format-date($today, '[DWw]'):
      Twenty Six

    The day in German words - format-date($today, '[Dw]', 'de', (), ()):
      sechsundzwanzig

    Using format-date($today, '[dwo]):
      It's the two hundred and seventh day of the year.

    Using format-date($today, '[d1o]):
      It's the 207th day of the year.

  The grand finale:
    Today is Wednesday, July 26, 2006,
      the 26th day of the 30th week of the year,
      the 4th week of the month,
      in the GMT-4 time zone.
```

Обратите внимание на использование кода языка `de` в двух примерах. С разными комбинациями модификаторов мы вывели компоненты даты в числовом и строковом виде, в порядковом и количественном форматах. Форматная строка сохраняется в переменной для повторного использования.

[2.0] format-dateTime()

Форматирует значение `xs:dateTime` в соответствии с форматной строкой.

Синтаксис

```
xs:string? format-dateTime(xs:dateTime?, $picture as xs:string)
xs:string? format-dateTime(xs:dateTime?, $picture as xs:string,
                           $language as xs:string?,
                           $calendar as xs:string?,
                           $country as xs:string?)
```

Ввод

В первой разновидности эта функция получает необязательное значение `xs:dateTime` и форматную строку. Если значение `xs:dateTime` не указано, используются текущие дата и время. Второй разновидности передаются строки, представляющие желательные коды языка, календаря и страны для выходного значения.

Далее перечислены коды, определенные для форматных строк:

`Y` Год. По умолчанию используется значение `[Y1]`, генерирующее год из четырех цифр.

`M` Месяц. По умолчанию используется значение `[M1]`, генерирующее номер месяца из одной или двух цифр.

`D` День месяца. По умолчанию используется значение `[D1]`, генерирующее номер дня из одной или двух цифр.

`d` День года. По умолчанию используется значение `[d1]`, генерирующее номер дня из одной, двух или трех цифр (например, для 26 июля невисокосного года генерируется значение `207`).

`F` День недели. По умолчанию используется значение `[Fn]`, генерирующее название дня недели в нижнем регистре (например, `wednesday`).

`W` Неделя года. По умолчанию используется значение `[W1]`, генерирующее номер недели в году из одной или двух цифр.

`w` Неделя месяца. По умолчанию используется значение `[w1]`, генерирующее номер недели в месяце из одной цифры.

`H` Час (по 24-часовой шкале). По умолчанию используется значение `[H1]`, генерирующее час из одной или двух цифр. Обратите внимание: сгенерированное значение никогда не превышает `23`.

`h` Час (по 12-часовой шкале). По умолчанию используется значение `[h1]`, генерирующее час из одной или двух цифр.

`P` Префикс половины суток (AM/PM). По умолчанию используется значение `[Pn]`, генерирующее префикс в нижнем регистре (`a.m.` или `p.m.`).

`m` Минуты. По умолчанию используется значение `[m01]`, генерирующее минуты из двух цифр.

`s` Секунды. По умолчанию используется значение `[s01]`, генерирующее секунды из двух цифр.

`f` Дробная часть секунд. По умолчанию используется значение `[f1]`, генерирующее дробную часть секунд с точностью до трех цифр.

`Z` Часовой пояс, заданный смещением от всемирного координированного времени UTC. По умолчанию используется значение `[Z1]`, генерирующее смещение от UTC в часах и минутах (например, `-04:00`).

`z` Часовой пояс, заданный смещением от гринвичского среднего времени GMT. По умолчанию используется значение `[z1]`, генерирующее смещение от GMT (например, `GMT-4`).

За полной информацией о форматных строках даты, времени и чисел обращайтесь к приложению F.

Вывод

Строка, представляющая значение `xs:dateTime`, отформатированное в соответствии с форматной строкой. Если аргумент `xs:dateTime` является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XSLT, раздел 16.5 «Formatting Dates and Times».

Пример

Следующая таблица стилей демонстрирует различные способы форматирования значения `xs:dateTime` и его компонентов:

```
<?xml version="1.0"?>
<!-- format-dateTime.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the format-dateTime() function:&#xA;</xsl:text>
```

```
    <xsl:variable name="today" select="current-dateTime()"/>
    <xsl:text>&#xA;  The current date and time is: </xsl:text>
    <xsl:value-of select="$today"/>
    <xsl:variable name="longFormat"
      select="concat('It''s [h1]:[m11] [P] on ',
              '[FNn], [MNn] [D1], [Y], &#xA;      ',
              'the [D1o] day of the [W1o] week of the year,',
              '&#xA;      the [w1o] week of the month,',
              '&#xA;      in the [zWw] time zone.')"/>

    <xsl:text>&#xA;&#xA;    The year - format-dateTime</xsl:text>
    <xsl:text>($today, '[Y]'): </xsl:text>
    <xsl:value-of select="format-dateTime($today, '[Y]')"/>
    <xsl:text>&#xA;&#xA;    The hour (12-hour clock) - </xsl:text>
    <xsl:text>format-dateTime($today, '[h] [PN]'): </xsl:text>
    <xsl:value-of select="format-dateTime($today, '[h] [PN]')"/>
    <xsl:text>&#xA;&#xA;    The minutes - format-dateTime(</xsl:text>
    <xsl:text>$today, '[m11]'): </xsl:text>
    <xsl:value-of select="format-dateTime($today, '[m11]')"/>
    <xsl:text>&#xA;&#xA;    The seconds - format-dateTime(</xsl:text>
    <xsl:text>$today, '[s11].[f1111]): </xsl:text>
    <xsl:value-of select="format-dateTime($today, '[s11].[f1111]')"/>
    <xsl:text>&#xA;&#xA;    The complete time - format-dateTime(</xsl:text>
    <xsl:text>$today, '[h]:[m11]:[s11] [P]'): &#xA;      </xsl:text>
    <xsl:value-of
      select="format-dateTime($today, '[h]:[m11]:[s11] [P]')"/>
    <xsl:text>&#xA;&#xA;    The day in words - format-dateTime(</xsl:text>
    <xsl:text>$today, '[FNn]'): &#xA;      </xsl:text>
    <xsl:value-of select="format-dateTime($today, '[FNn]')"/>

    <xsl:text>&#xA;&#xA;  The grand finale:</xsl:text>
    <xsl:text>&#xA;    </xsl:text>
    <xsl:value-of select="format-dateTime($today, $longFormat)"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the format-dateTime() function:

  The current date and time is: 2006-07-26T13:58:25.790-04:00

    The year - format-dateTime($today, '[Y]'): 2006

    The hour (12-hour clock) - format-dateTime($today, '[h] [PN]'): 1 P.M.

    The minutes - format-dateTime($today, '[m11]'): 58

    The seconds - format-dateTime($today, '[s11].[f1111]): 25.7900

    The complete time - format-dateTime($today, '[h]:[m11]:[s11] [P]'):
      1:58:25 p.m.

    The day in words - format-dateTime($today, '[FNn]'):
      Wednesday

  The grand finale:
    It's 1:58 p.m. on Wednesday, July 26, 2006,
      the 26th day of the 30th week of the year,
      the 4th week of the month,
      in the GMT-4 time zone.
```

format-number()

Получает число и форматирует его в виде строки.

Синтаксис

```
[1.0] string format-number($value as number, $picture as string,
                           $formatName as string?)
[2.0] xs:string format-number($value as numeric, $picture as xs:string,
                              $formatName as xs:string?)
```

Ввод

Форматируемое число и форматная строка являются обязательными аргументами. Третий (необязательный) аргумент содержит имя числового формата; если третий аргумент не задан, используется формат по умолчанию.

[2.0] Числовой аргумент должен относиться к одному из четырех числовых типов (`xs:float`, `xs:decimal`, `xs:double` или `xs:integer`). Если он не относится к числовому типу или не может быть преобразован к нему, процессор XSLT выдает ошибку. Процессор XSLT также выдает ошибку, если числовой формат с указанным именем не существует.

Вывод

Число, отформатированное по правилам, заданным другими аргументами. Во втором аргументе используются следующие специальные символы:

`#` Представляет цифру. Начальные и завершающие нули не отображаются. При форматировании числа `4.0` по строке `#.##` будет получена строка `4`.

`0` Представляет цифру. В отличие от символа `#`, начальные и конечные нули всегда отображаются. При форматировании числа `4.1` по строке `#.00` будет получена строка `4.10`.

`.` Представляет десятичную точку.

`-` Представляет знак «минус».

`,` Представляет разделитель групп разрядов.

`;` Отделяет шаблон положительного числа от шаблона отрицательного числа.

`%` Указывает, что число должно отображаться в процентах (значение умножается на 100, а затем отображается со знаком %). При форматировании числа `.76` по строке `##%` будет получена строка `76%`.

`\u2030` Символ Юникода для знака промилле (значение умножается на 1000, а затем отображается со знаком промилле). При форматировании числа `.768` по строке `###\u2030` будет получена строка `768‰`.

За полной информацией о форматных строках даты, времени и чисел обращайтесь к приложению F.

Третий аргумент, если он задан, должен содержать имя элемента `<xsl:decimal-format>`. Элемент `<xsl:decimal-format>` позволяет определить символ, используемый для отделения дробной части; разделитель групп разрядов; строку, используемую для обозначения бесконечности; а также задать другие параметры форматирования. За дополнительной информацией обращайтесь к описанию элемента `<xsl:decimal-format>` в приложении A.

Где определяется

[1.0] XSLT, раздел 12.3 «Number Formatting».

[2.0] XSLT, раздел 16.4 «Number Formatting».

Пример

Следующая таблица стилей демонстрирует различные варианты использования функции `format-number()`:

```
<?xml version="1.0"?>
<!-- format-number.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:decimal-format name="f1"
    decimal-separator=":"
    grouping-separator="/"/>

  <xsl:decimal-format name="f2"
    infinity="Really, really big"
    NaN="[not a number]"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the format-number() </xsl:text>
    <xsl:text>function:&#xA;&#xA;</xsl:text>

    <xsl:text>  1. format-number(528.3, '#.#;-#.#')=</xsl:text>
    <xsl:value-of
      select="format-number(528.3, '#.#;-#.#')"/>
    <xsl:text>&#xA;  2. format-number(528.3, </xsl:text>
    <xsl:text>'0,000.00;-0,000.00')=</xsl:text>
    <xsl:value-of
      select="format-number(528.3, '0,000.00;-0,000.00')"/>
    <xsl:text>&#xA;  3. format-number(-23528.3, </xsl:text>
    <xsl:text>'$#,###.00;($#,###.00)')=</xsl:text>
    <xsl:value-of
      select="format-number(-23528.3, '$#,###.00;($#,###.00)')"/>
    <xsl:text>&#xA;  4. format-number(1528.3, </xsl:text>
    <xsl:text>'#/###:00', 'f1')=</xsl:text>
    <xsl:value-of
      select="format-number(1528.3, '#/###:00;-#/###:00', 'f1')"/>
    <xsl:text>&#xA;  5. format-number(1 div 0, </xsl:text>
    <xsl:text>'###,###.00', 'f2')=</xsl:text>
    <xsl:value-of
      select="format-number(1 div 0, '###,###.00', 'f2')"/>
    <xsl:text>&#xA;  6. format-number(blue div orange, </xsl:text>
    <xsl:text>'#.##', 'f2')=</xsl:text>
    <xsl:value-of
      select="format-number(blue div orange, '#.##', 'f2')"/>
    <xsl:text>&#xA;&#xA;  </xsl:text>

    <xsl:value-of select="report/title"/>
    <xsl:text> - </xsl:text>
    <xsl:value-of select="report/@month"/>
    <xsl:text>/</xsl:text>
    <xsl:value-of select="report/@year"/>
```

```
    <xsl:text>&#xA;&#xA;</xsl:text>

    <xsl:variable name="totalSales"
      select="sum(/report/brand/units)"/>

    <xsl:for-each select="report/brand">
      <xsl:text>    </xsl:text>
      <xsl:value-of select="name"/>
      <xsl:text> - </xsl:text>
      <xsl:value-of select="format-number(units, '##,###')"/>
      <xsl:text> bars sold, </xsl:text>
      <xsl:value-of
        select="format-number(units div $totalSales, '##%')"/>
      <xsl:text> of all sales.&#xA;</xsl:text>
    </xsl:for-each>

    <xsl:text>&#xA;  Total sales: </xsl:text>
    <xsl:value-of
      select="format-number(sum(/report/brand/units), '##,###')"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей используется со следующим документом:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  ...
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Результат применения таблицы стилей:

```
Tests of the format-number() function:

  1. format-number(528.3, '#.#;-#.#')=528.3
  2. format-number(528.3, '0,000.00;-0,000.00')=0,528.30
  3. format-number(-23528.3, '$#,###.00;($#,###.00)')=($23,528.30)
  4. format-number(1528.3, '#/###:00', 'f1')=1/528:30
  5. format-number(1 div 0, '###,###.00', 'f2')=Really, really big
  6. format-number(blue div orange, '#.##', 'f2')=[not a number]

  Chocolate bar sales - 8/2006

    Lindt - 27,408 bars sold, 30% of all sales.
    Callebaut - 8,203 bars sold, 9% of all sales.
    Valrhona - 22,101 bars sold, 24% of all sales.
    Perugina - 14,336 bars sold, 16% of all sales.
    Ghirardelli - 19,268 bars sold, 21% of all sales.

  Total sales: 91,316
```

В первых трех примерах функция format-number() управляет форматированием числа (отделением дробной части, разделителями разрядов, форматом отрицательных чисел). В четвертом примере используется элемент <decimal-format> с именем f1, определяющий косую черту (/) в качестве разделителя групп разрядов и двоеточие (:) в качестве разделителя дробной части.

Примеры 5 и 6 создают проблемы в XSLT 2.0. Пример 5 не работает, потому что деление на нуль является фатальной ошибкой для значений xs:integer в XSLT 2.0. Пример 6 не работает, потому что оператору div необходимы числовые операнды. Чтобы использовать обработку по правилам XSLT 1.0 в таблицах стилей XSLT 2.0, включите атрибут version="1.0" в соответствующий элемент <xsl:value-of>:

```
<xsl:value-of version="1.0"
  select="format-number(1 div 0, '###,###.00', 'f2')"/>
```

Примеры последней группы извлекают данные из документа с информацией о продажах шоколада и форматируют общий объем продаж по каждому виду шоколада. Таблица стилей выдает также проценты от объема продаж по каждому бренду и выводит общий объем продаж.

[2.0] format-time()

Форматирует значение xs:time в соответствии с форматной строкой.

Синтаксис

```
xs:string format-time(xs:time?, $picture as xs:string)
xs:string format-time(xs:time?, $picture as xs:string,
                      $language as xs:string?,
                      $calendar as xs:string?,
                      $country as xs:string?)
```

Ввод

В первой разновидности эта функция получает необязательное значение xs:time и форматную строку. Если значение xs:time не указано, используется текущее время. Второй разновидности передаются строки, представляющие желательные коды языка, календаря и страны для выходного значения.

Далее перечислены коды, определенные для форматных строк:

H — Час (по 24-часовой шкале). По умолчанию используется значение [H1], генерирующее час из одной или двух цифр. Обратите внимание: сгенерированное значение никогда не превышает 23.

h — Час (по 12-часовой шкале). По умолчанию используется значение [h1], генерирующее час из одной или двух цифр.

P — Префикс половины суток (AM/PM). По умолчанию используется значение [Pn], генерирующее префикс в нижнем регистре (a.m. или p.m.).

m — Минуты. По умолчанию используется значение [m01], генерирующее минуты из двух цифр.

s — Секунды. По умолчанию используется значение [s01], генерирующее секунды из двух цифр.

f — Дробная часть секунд. По умолчанию используется значение [f1], генерирующее дробную часть секунд с точностью до трех цифр.

`Z` Часовой пояс, заданный смещением от всемирного координированного времени UTC. По умолчанию используется значение `[Z1]`, генерирующее смещение от UTC в часах и минутах (например, `-04:00`).

`z` Часовой пояс, заданный смещением от гринвичского среднего времени GMT. По умолчанию используется значение `[z1]`, генерирующее смещение от GMT (например, `GMT-4`).

За полной информацией о форматных строках даты, времени и чисел обращайтесь к приложению F.

Вывод

Строка, представляющая значение `xs:time`, отформатированное в соответствии с форматной строкой. Если аргумент `xs:time` является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XSLT, раздел 16.5 «Formatting Dates and Times».

Пример

Следующая таблица стилей демонстрирует различные варианты использования функции `format-time()`:

```
<?xml version="1.0"?>
<!-- format-time.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the format-time() function:&#xA;</xsl:text>

    <xsl:variable name="now" select="current-time()"/>
    <xsl:text>&#xA;  The current date and time is: </xsl:text>
    <xsl:value-of select="$now"/>
    <xsl:variable name="longFormat"
      select="concat('It''s [h1]:[m11]:[s11] [P] ',
              'in the [zWw] time zone.')"/>

    <xsl:text>&#xA;&#xA;    The hour (24-hour clock) - </xsl:text>
    <xsl:text>format-time($now, '[H]'): </xsl:text>
    <xsl:value-of select="format-time($now, '[H]')"/>
    <xsl:text>&#xA;&#xA;    The minutes - format-time(</xsl:text>
    <xsl:text>$now, '[m11]'): </xsl:text>
    <xsl:value-of select="format-time($now, '[m11]')"/>
    <xsl:text>&#xA;&#xA;    Using format-time(</xsl:text>
    <xsl:text>$now, '[mWwo]'): &#xA;      It's the </xsl:text>
    <xsl:value-of select="format-time($now, '[mWwo]')"/>
    <xsl:text> minute of the hour.</xsl:text>
    <xsl:text>&#xA;&#xA;    The time - format-time(</xsl:text>
    <xsl:text>$now, '[H]:[m11]'): &#xA;      </xsl:text>
    <xsl:value-of
      select="format-time($now, '[H]:[m11]')"/>

    <xsl:text>&#xA;&#xA;  The grand finale:</xsl:text>
```

```
      <xsl:text>&#xA;    </xsl:text>
      <xsl:value-of select="format-time($now, $longFormat)"/>
    </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the format-time() function:

  The current date and time is: 14:10:00.028-04:00

    The hour (24-hour clock) - format-time($now, '[H]'): 14

    The minutes - format-time($now, '[m11]'): 10

    Using format-time($now, '[mWwo]'):
      It's the Tenth minute of the hour.

    The time - format-time($now, '[H]:[m11]'):
      14:10

  The grand finale:
    It's 2:10:00 p.m. in the GMT-4 time zone.
```

function-available()

Проверяет доступность заданной функции для процессора XSLT. Функция позволяет строить таблицы стилей, корректно реагирующие на недоступность некоторой расширяющей функции.

Синтаксис

```
[1.0] boolean function-available($functionName as string)
[2.0] xs:boolean function-available($functionName as xs:string,
                                    $arity as xs:integer?)
```

Ввод

Имя функции. Обычно имя уточняется пространством имен; если пространство имен для имени функции не пусто, то функция является функцией расширения. В противном случае предполагается, что функция определяется в спецификации XSLT или XPath.

[2.0] Версия этой функции для XSLT 2.0 также может вызываться с аргументом, определяющим количество параметров («арность» функции). При помощи этого аргумента можно указать, сколько параметров принимает функция. Например, вызов `function-available('ora:greatFunction',2)` может вернуть `true`, тогда как вызов `function-available('ora:greatFunction', 3)` вернет `false`. При использовании вызова `function-available('me:my-function', 2)` в процессоре XSLT 1.0 произойдет ошибка времени выполнения.

Вывод

Логическое значение `true`, если функция доступна; `false` в противном случае.

Где определяется

[1.0] XSLT, раздел 15 «Fallback».

[2.0] XSLT, раздел 18.1 «Extension Functions».

Пример

Для демонстрации работы функции `function-available()` **будет использоваться следующий документ XML:**

```
<?xml version="1.0"?>
<!-- favorites.xml -->
<list>
  <title>A few of my favorite albums</title>
  <listitem>A Love Supreme</listitem>
  <listitem>Beat Crazy</listitem>
  <listitem>Here Come the Warm Jets</listitem>
  <listitem>Kind of Blue</listitem>
  <listitem>London Calling</listitem>
  <listitem>Remain in Light</listitem>
  <listitem>The Joshua Tree</listitem>
  <listitem>The Indestructible Beat of Soweto</listitem>
</list>
```

Таблица стилей:

```
<?xml version="1.0"?>
<!-- function-available.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:jpeg="class:JPEGWriter"
  extension-element-prefixes="jpeg">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of function-avilable():&#xA;&#xA;</xsl:text>
    <xsl:for-each select="list/listitem">
      <xsl:choose>
        <xsl:when test="function-available('jpeg:buildJPEGFile')">
          <xsl:value-of
            select="jpeg:buildJPEGFile(., 'titlebar.jpg',
            concat('album', position(), '.jpg'),
            'Verdana Bold Italic', 20, 12, 20, '000000')"/>
          <xsl:text>See the file </xsl:text>
          <xsl:value-of select="concat('album', position(), '.jpg')"/>
          <xsl:text> to see the title of album #</xsl:text>
          <xsl:value-of select="position()"/>
          <xsl:text>&#xA;</xsl:text>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="position()"/>
          <xsl:text>. </xsl:text>
          <xsl:value-of select="."/>
          <xsl:text>&#xA;</xsl:text>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Учтите, что механизм определения и использования функций расширения зависит от конкретного процессора. В приведенном примере используется специфический механизм Xalan-J: мы создаем новое пространство имен (`jpeg`) и определяем его URI-адрес в виде `class:JPEGWriter`. При вызове функции, проверяющей доступность пространства имен, Xalan-J пытается загрузить класс Java с именем `JPEGWriter`. Если этот класс не найден, `function-available()` возвращает `false`. Если поиск оказывается успешным, Xalan-J анализирует класс и возвращает `true` в том случае, если класс Java содержит метод с указанным именем (`buildJPEGFile` в данном примере).

Учтите также, что функция `function-available()` не обязана проверять сигнатуру метода. В данном примере, если Xalan-J находит класс и метод с указанным именем, функция возвращает `true`. Это не означает, что Xalan-J проверил количество и тип аргументов, передаваемых функции, и убедился в их правильности. Если данные неверны, произойдет ошибка времени выполнения.

Если функция `buildJPEGFile()` доступна, она вызывается для создания JPEG-файлов с названиями всех любимых альбомов. Если функция недоступна, названия просто выводятся в выходной поток. Результаты при доступной функции `buildJPEGFile()`:

```
See the file album1.jpg to see the title of album #1
See the file album2.jpg to see the title of album #2
See the file album3.jpg to see the title of album #3
See the file album4.jpg to see the title of album #4
See the file album5.jpg to see the title of album #5
See the file album6.jpg to see the title of album #6
See the file album7.jpg to see the title of album #7
See the file album8.jpg to see the title of album #8
```

Все названия альбомов (текст элементов `<listitem>`) преобразуются в графику JPEG. Например, файл *album8.jpg* выглядит так, как показано на рис. С.3.

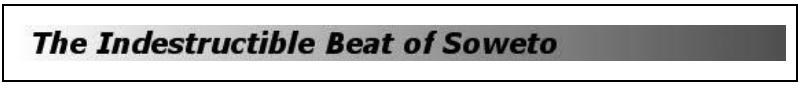


Рис. С.3. *Графика, сгенерированная для восьмого элемента <listitem>*

Если процессор XSLT не может загрузить функцию расширения (возможно, вы используете процессор Saxon, который не загружает классы расширения подобным образом; возможно, файл *.class* не включен в путь поиска; возможно, имя функции содержит ошибку, и т. д.), результат будет выглядеть так:

```
1. A Love Supreme
2. Beat Crazy
3. Here Come the Warm Jets
4. Kind of Blue
5. London Calling
6. Remain in Light
7. The Joshua Tree
8. The Indestructible Beat of Soweto
```

generate-id()

Генерирует уникальный идентификатор (имя XML) для заданного узла. Если набор узлов не задан, `generate-id()` генерирует идентификатор для контекстного узла.

Синтаксис

```
[1.0] string generate-id(node-set?)
[2.0] xs:string generate-id()
[2.0] xs:string generate-id(node()?)
```

Ввод

[1.0] Необязательный набор узлов. Если набор узлов не задан, функция генерирует идентификатор для контекстного узла. Для пустого набора узлов `generate-id()` возвращает пустую строку.

[2.0] Необязательный узел. Если узел не задан, функция генерирует идентификатор для контекстного узла. Если аргумент является пустой последовательностью, результат представляет собой строку нулевой длины.

В XSLT 1.0 при передаче набора узлов функция `generate-id()` генерировала уникальный идентификатор для первого объекта в наборе узлов; все остальные узлы при этом игнорировались. *В XSLT 2.0 передача* `generate-id()` *последовательности, содержащей более одного узла, приводит к ошибке.*

Вывод

Уникальный идентификатор или пустая строка (при передаче пустого набора узлов). Несколько важных особенностей функции `generate-id()`, о которых необходимо знать:

- В рамках конкретного преобразования каждый раз, когда функция `generate-id()` вызывается для заданного узла, она возвращает один и тот же идентификатор. Если запустить преобразование на следующий день, нет гарантий, что `generate-id()` выдаст тот же идентификатор, что и сегодня. Все завтрашние вызовы `generate-id()` для заданного узла (в рамках одного преобразования) будут генерировать один и тот же идентификатор, но он не будет обязан совпадать с тем, который был сгенерирован сегодня.
- Функция `generate-id()` не обязана проверять, не совпадает ли сгенерированный идентификатор с другим идентификатором, уже присутствующим в документе. Другими словами, если элемент в вашем документе содержит атрибут `ID` со значением `sdk3829a`, не исключено, что генерируемое `generate-id()` значение окажется равным `sdk3829a`. Это маловероятно, но теоретически возможно.
- При вызове `generate-id()` для двух разных узлов будут сгенерированы два разных идентификатора.
- *[1.0]* Для набора узлов `generate-id()` возвращает идентификатор узла, следующего первым в порядке документа.

  [2.0] В XSLT 2.0 передача `generate-id()` последовательности, содержащей более одного объекта, приводит к фатальной ошибке.
- Если функции передается пустой набор узлов (скажем, при вызове `generate-id(fleeber)`, когда в текущем контексте нет ни одного элемента `<fleeber>`), `generate-id()` возвращает пустую строку.

Где определяется

[1.0] XSLT, раздел 12.4 «Miscellaneous Additional Functions».

[2.0] XSLT, раздел 16.6 «Miscellaneous Additional Functions».

Пример

Отчет о продажах шоколада будет использован в качестве входных данных для нашей таблицы стилей:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Таблица стилей генерирует новый идентификатор для каждого узла входного документа. Просто для того, чтобы быть уверенными, что процессор XSLT генерирует одинаковые значения, мы повторно генерируем новый идентификатор для каждого узла. Два сгенерированных идентификатора должны совпадать:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the generate-id() </xsl:text>
    <xsl:text>function:&#xA;&#xA;</xsl:text>

    <xsl:text>  We'll generate IDs for every node </xsl:text>
    <xsl:text>in the XML source:&#xA;&#xA;</xsl:text>
    <xsl:for-each select="//*">
      <xsl:text>    Node name: &lt;</xsl:text>
      <xsl:value-of select="name()"/>
      <xsl:text>&gt; - generated id: </xsl:text>
      <xsl:value-of select="generate-id()"/>
```

```
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>

    <xsl:text>&#xA;  Now we'll try it again...&#xA;&#xA;</xsl:text>
    <xsl:for-each select="//*">
      <xsl:text>    Node name: &lt;</xsl:text>
      <xsl:value-of select="name()"/>
      <xsl:text>&gt; - generated id: </xsl:text>
      <xsl:value-of select="generate-id()"/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
A test of the generate-id() function:

  We'll generate IDs for every node in the XML source:

    Node name: <report> - generated id: d1e2
    Node name: <title> - generated id: d1e4
    Node name: <brand> - generated id: d1e7
    Node name: <name> - generated id: d1e9
    Node name: <units> - generated id: d1e12
    Node name: <brand> - generated id: d1e16
    Node name: <name> - generated id: d1e18
    Node name: <units> - generated id: d1e21
    Node name: <brand> - generated id: d1e25
    Node name: <name> - generated id: d1e27
    Node name: <units> - generated id: d1e30
    Node name: <brand> - generated id: d1e34
    Node name: <name> - generated id: d1e36
    Node name: <units> - generated id: d1e39
    Node name: <brand> - generated id: d1e44
    Node name: <name> - generated id: d1e46
    Node name: <units> - generated id: d1e49

  Now we'll try it again...

    Node name: <report> - generated id: d1e2
    Node name: <title> - generated id: d1e4
    Node name: <brand> - generated id: d1e7
    Node name: <name> - generated id: d1e9
    Node name: <units> - generated id: d1e12
    Node name: <brand> - generated id: d1e16
    Node name: <name> - generated id: d1e18
    Node name: <units> - generated id: d1e21
    Node name: <brand> - generated id: d1e25
    Node name: <name> - generated id: d1e27
    Node name: <units> - generated id: d1e30
    Node name: <brand> - generated id: d1e34
    Node name: <name> - generated id: d1e36
    Node name: <units> - generated id: d1e39
    Node name: <brand> - generated id: d1e44
    Node name: <name> - generated id: d1e46
    Node name: <units> - generated id: d1e49
```

Идентификаторы, сгенерированные при двух разных попытках, совпадают. Учтите, что другие процессоры генерируют другие значения. У Xalan-J первое сгенерированное значение было равно `N10002`, а у AltovaXML – `idreport220739936`.

[2.0] hours-from-dateTime()

Для заданного значения `xs:dateTime` возвращает часы.

Синтаксис

```
xs:integer? hours-from-dateTime(xs:dateTime?)
```

Ввод

Значение `xs:dateTime`.

Вывод

Значение `xs:integer`, представляющее компонент часов заданного значения `xs:dateTime`. Если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Таблица стилей для демонстрации работы функции `hours-from-dateTime()`:

```
<?xml version="1.0"?>
<!-- hours-from-datetime.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the hour from an xs:dateTime:</xsl:text>
    <xsl:variable name="currentDateTime" as="xs:dateTime"
      select="current-dateTime()"/>
    <xsl:text>&#xA;&#xA;  The current date and time is: </xsl:text>
    <xsl:value-of select="$currentDateTime"/>

    <xsl:text>&#xA;&#xA;  The current hour is: </xsl:text>
    <xsl:value-of select="hours-from-dateTime($currentDateTime)"/>
    <xsl:text>&#xA;    Also known as </xsl:text>
    <xsl:value-of select="format-dateTime($currentDateTime, '[h] [Pn]')"/>
    <xsl:text>&#xA;    or </xsl:text>
    <xsl:value-of select="format-dateTime($currentDateTime, '[H] [ZN]')"/>
    <xsl:text>&#xA;    or </xsl:text>
    <xsl:value-of select="format-dateTime($currentDateTime, '[H] [z]')"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Extracting the hour from an xs:dateTime:

  The current date and time is: 2006-11-16T04:17:15.778-05:00

  The current hour is: 4
    Also known as 4 a.m.
    or 4 EST
    or 4 GMT-5
```

Одни результаты сгенерированы функцией `hours-from-dateTime()`, другие – функцией `format-dateTime()` с форматной строкой, выбирающей только компонент часов из значения `xs:dateTime`.

См. также

Определения функций *[2.0]* `day-from-dateTime()`, *[2.0]* `format-dateTime()`, *[2.0]* `minutes-from-dateTime()`, *[2.0]* `month-from-dateTime()`, *[2.0]* `seconds-from-dateTime()`, *[2.0]* `timezone-from-dateTime()` и *[2.0]* `year-from-dateTime()`.

[2.0] hours-from-duration()

Возвращает количество часов во временном промежутке `xs:duration`.

Синтаксис

```
xs:integer? hours-from-duration(xs:duration?)
```

Ввод

Значение `xs:duration`.

Вывод

Значение `xs:integer`, представляющее компонент часов в заданном промежутке `xs:duration`. Учтите, что для `xs:yearMonthDuration` эта функция всегда возвращает `0`, потому что `xs:yearMonthDuration` не содержит компонент часов. Кроме того, если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Следующая таблица стилей демонстрирует работу функции `hours-from-duration()` для всех трех типов промежутков:

```
<?xml version="1.0"?>
<!-- hours-from-duration.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>
```

```
  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the hours component from durations:</xsl:text>

    <xsl:variable name="sampleDuration" as="xs:duration"
      select="xs:duration('P3Y8M2DT4H23M12.2S')"/>
    <xsl:variable name="sampleYearMonthDuration" as="xs:yearMonthDuration"
      select="xs:yearMonthDuration('P3Y8M')"/>
    <xsl:variable name="sampleDayTimeDuration" as="xs:dayTimeDuration"
      select="xs:dayTimeDuration('P2DT4H23M12.2S')"/>

    <xsl:text>&#xA;&#xA;  A sample xs:duration: </xsl:text>
    <xsl:value-of select="$sampleDuration"/>
    <xsl:text>&#xA;    The hours component of this duration is </xsl:text>
    <xsl:value-of select="hours-from-duration($sampleDuration)"/>
    <xsl:text>.</xsl:text>

    <xsl:text>&#xA;&#xA;  A sample xs:yearMonthDuration: </xsl:text>
    <xsl:value-of select="$sampleYearMonthDuration"/>
    <xsl:text>&#xA;    The hours component of this duration is </xsl:text>
    <xsl:value-of select="hours-from-duration($sampleYearMonthDuration)"/>
    <xsl:text>.</xsl:text>

    <xsl:text>&#xA;&#xA;  A sample xs:dayTimeDuration: </xsl:text>
    <xsl:value-of select="$sampleDayTimeDuration"/>
    <xsl:text>&#xA;    The hours component of this duration is </xsl:text>
    <xsl:value-of select="hours-from-duration($sampleDayTimeDuration)"/>
    <xsl:text>.</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Extracting the hours component from durations:

  A sample xs:duration: P3Y8M2DT4H23M12.2S
    The hours component of this duration is 4.

  A sample xs:yearMonthDuration: P3Y8M
    The hours component of this duration is 0.

  A sample xs:dayTimeDuration: P2DT4H23M12.2S
    The hours component of this duration is 4.
```

Обратите внимание: вызов `hours-from-duration()` **для значения** `xs:yearMonthDuration` **возвращает** `0`**.**

См. также

Определения функций ***[2.0]*** `days-from-duration()`**,** ***[2.0]*** `minutes-from-duration()`**,** ***[2.0]*** `months-from-duration()`**,** ***[2.0]*** `seconds-from-duration()` **и** ***[2.0]*** `years-from-duration()`**.**

[2.0] hours-from-time()

Для заданного значения `xs:time` **возвращает часы.**

Синтаксис

```
xs:integer? hours-from-time (xs:time?)
```

Ввод

Значение `xs:time`.

Вывод

Значение `xs:integer`, представляющее компонент часов заданного значения `xs:time`. Если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Таблица стилей для демонстрации работы функции `hours-from-time()`:

```
<?xml version="1.0"?>
<!-- hours-from-time.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the hours component from an xs:time:</xsl:text>
    <xsl:variable name="currentTime" as="xs:time" select="current-time()"/>
    <xsl:text>&#xA;&#xA;  The current time is: </xsl:text>
    <xsl:value-of select="$currentTime"/>

    <xsl:text>&#xA;&#xA;  The current hour is: </xsl:text>
    <xsl:value-of select="hours-from-time($currentTime)"/>
    <xsl:text>&#xA;    Also known as </xsl:text>
    <xsl:value-of select="format-time($currentTime, '[h] [Pn]')"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="format-time($currentTime, '[H] [ZN]')"/>
    <xsl:text>, or </xsl:text>
    <xsl:value-of select="format-time($currentTime, '[H] [z]')"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Extracting the hours component from an xs:time:

  The current time is: 17:02:52.515-05:00

  The current hour is: 17
    Also known as 5 p.m., 17 EST, or 17 GMT-5
```

Первый результат сгенерирован функцией `hours-from-time()`, остальные – функцией `format-time()` с форматной строкой, выбирающей компонент часов из значения `xs:time`.

См. также

Определения функций *[2.0]* `format-time()`, *[2.0]* `minutes-from-time()`, *[2.0]* `seconds-from-time()` и *[2.0]* `timezone-from-time()`.

id()

Возвращает узел исходного дерева, у которого атрибут `ID` совпадает со значением(-ями), переданным(и) во входных данных.

Синтаксис

```
[1.0] node-set id(object)
[2.0] element()* id(xs:string*)
[2.0] element()* id(xs:string*, node()?)
```

Ввод

[1.0] Объект. Если входной объект является набором узлов, то результат представляет собой набор узлов, полученный в результате применения функции `id()` к строковому значению каждого узла в наборе узлов аргумента. Обычно аргумент относится к некоторому типу узла, который является строкой (или преобразуется в нее). Эта строка используется в качестве критерия поиска по всем атрибутам типа `ID`.

[2.0] Значение `xs:string` и необязательный узел. Если необязательный аргумент-узел присутствует, процессор ищет значение в документе, содержащем этот узел.

В XSLT 1.0 и 2.0, если строка содержит несколько значений, разделенных пробелами, каждое такое значение используется в качестве аргумента поиска функции `id()`. В приведенных примерах используются как простые строки, так и строки с несколькими значениями.

Помните об ограничении, установленном для типа данных XML `ID`: все атрибуты, объявленные с типом `ID`, входят в единый набор имен. Функция XSLT `key()` и связанный с ней элемент `<xsl:key>` помогают преодолеть это и другие ограничения; за дополнительной информацией обращайтесь к описанию функции `key()` в этом приложении, а также к описанию элемента `<xsl:key>` в приложении A.

Вывод

[1.0] Набор узлов, содержащий все узлы, у которых значение атрибута типа `ID` совпадает со строковыми значениями из входного набора узлов.

[2.0] Последовательность, содержащая все узлы, у которых значение `ID` равно одному из значений, заданных в строке поиска.

В действительном документе XML для каждого искомого значения будет найден только один узел; значение атрибута `ID` должно быть уникально среди всех атрибутов `ID` в документе. Поскольку процессор XSLT не обязан проверять действительность документа XML, набор узлов или последовательность может содержать более одного совпадения для каждого искомого значения.

Где определяется

[1.0] XPath, раздел 4.1 «Node Set Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.5 «Functions and Operators that Generate Sequences».

Пример

В следующем примере используется список компонентов и деталей со встроенным описанием DTD. Мы перечислим все компоненты, указывая после каждого из них названия всех деталей, используемых данным компонентом. Функция `id()` позволяет загрузить элемент `<part>` с заданным идентификатором; из него берется название детали.

Данный пример является полной противоположностью примера для функции XSLT 2.0 `idref()`. Здесь мы перебираем элементы `<part>` и перечисляем все компоненты, содержащие ссылку на текущую деталь.

Документ XML:

```
<?xml version="1.0"?>
<!-- parts-list1.xml -->
<!DOCTYPE parts-list [
<!ELEMENT parts-list       (component+, part+)>

<!ELEMENT component        (name, partref+)>
<!ATTLIST component        component-id ID #REQUIRED>

<!ELEMENT name             (#PCDATA)>

<!ELEMENT partref          EMPTY>
<!ATTLIST partref          refid IDREF #REQUIRED>

<!ELEMENT part             (name)>
<!ATTLIST part             part-id ID #REQUIRED
                           price CDATA #REQUIRED
                           source CDATA #REQUIRED>
]>

<parts-list>
  <component component-id="C28392-33-TT">
    <name>Turnip Twaddler</name>
    <partref refid="P81952-26-PK"/>
    <partref refid="P86679-52-SP"/>
    <partref refid="P81472-68-FD"/>
    <partref refid="P88107-39-GT"/>
  </component>
...
  <component component-id="C28772-63-OB">
    <name>Olive Bruiser</name>
    <partref refid="P80228-21-PT"/>
    <partref refid="P82387-85-PA"/>
  </component>

  <part part-id="P80228-21-PT" price="3.28" source="us">
    <name>Pitter</name>
  </part>
...
  <part part-id="P86994-25-RC" price="4.28" source="gb">
    <name>Ribbon Curler</name>
  </part>
</parts-list>
```

Для разрешения ссылок будет использоваться следующая таблица стилей:

```
<?xml version="1.0"?>
<!-- id.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Here is a test of the id() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:for-each select="/parts-list/component">
      <xsl:sort select="name"/>
      <xsl:text>&#xA;  </xsl:text>
      <xsl:value-of select="name"/>
      <xsl:text> (part #</xsl:text>
      <xsl:value-of select="@component-id"/>
      <xsl:text>) uses these parts:&#xA;    </xsl:text>
      <xsl:for-each select="id(partref/@refid)">
        <xsl:value-of select="name"/>
        <xsl:text>&#xA;    </xsl:text>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Here is a test of the id() function:

  Clam Teaser (part #C28100-38-CT) uses these parts:
    Spanner
    Feather Duster
    Ribbon Curler

  Cucumber Decorating Kit (part #C23813-03-CDK) uses these parts:
    Feather Duster
    Paring Knife
    Mucilage
    Ribbon Curler

  Lemon Snubber (part #C21630-29-LS) uses these parts:
    Spanner
    Grommet
    Paring Knife

  Olive Bruiser (part #C28772-63-OB) uses these parts:
    Pitter
    Patter

  Prawn Goader (part #C28813-70-PG) uses these parts:
    Paring Knife
    Mucilage
    Ribbon Curler

  Turnip Twaddler (part #C28392-33-TT) uses these parts:
    Spanner
    Feather Duster
    Grommet
    Paring Knife
```

Если аргумент функции id() является строкой, каждая отделенная пробелами лексема в строке считается отдельным идентификатором. Это означает, что id() можно также использовать с типом данных IDREFS. Измененная версия исходного документа с использованием атрибута IDREFS:

```
<?xml version="1.0"?>
<!-- parts-list2.xml -->
<!DOCTYPE parts-list [
<!ELEMENT parts-list       (component+, part+)>

<!ELEMENT component        (name, partref)>
<!ATTLIST component        component-id ID #REQUIRED>

<!ELEMENT name             (#PCDATA)>

<!ELEMENT partref          EMPTY>
<!ATTLIST partref          refid IDREFS #REQUIRED>

<parts-list>
  <component component-id="C28392-33-TT">
    <name>Turnip Twaddler</name>
    <partref
      refid="P81952-26-PK P86679-52-SP P81472-68-FD P88107-39-GT"/>
  </component>
...
  <part part-id="P80228-21-PT" price="3.28" source="us">
    <name>Pitter</name>
  </part>
...
</parts-list>
```

Изменение только одно: атрибут refid элемента <partref> теперь относится к типу данных IDREFS. Несмотря на это изменение, мы можем преобразовать файл *parts-list2.xsl* с той же таблицей стилей и получить те же результаты.

Значение любого атрибута типа ID, IDREF или IDREFS должно быть действительным именем XML. Скажем, значение 86679-52-SP *не* является действительным именем XML, поэтому процессор XSLT, проверяющий исходный документ XML, может не сгенерировать результаты для этого конкретного значения. Чтобы понять, какие проблемы при этом могут возникнуть, измените все значения ID так, чтобы они начинались с цифры, и обработайте документ процессором XSLT Saxon. Результат выглядит примерно так:

```
Here is a test of the id() function:

  Turnip Twaddler (part #28392-33-TT) uses these parts:

  Prawn Goader (part #28813-70-PG) uses these parts:

  Clam Teaser (part #28100-38-CT) uses these parts:

  Cucumber Decorating Kit (part #23813-03-CDK) uses these parts:

  Lemon Snubber (part #21630-29-LS) uses these parts:

  Olive Bruiser (part #28772-63-OB) uses these parts:
```

При преобразовании документа в Saxon никакие результаты не найдены. Запустив обработку в Saxon с флагом -v для проверки действитель-

ности исходного документа XML, вы получите десятки сообщений следующего вида:

```
Error on line 59 column 56 of file:/C:/XSLT20/AppendixC/parts-list3.xml:
  SXXP0003: Error reported by XML parser: Attribute value "80228-21-PT"
of type ID must be an NCName when namespaces are enabled.
```

Отредактируйте исходный документ XML так, чтобы все идентификаторы представляли собой действительные имена XML – и вы получите ожидаемые результаты.

Помните об этом при работе с функцией `id()`*; это коварная ошибка, которую трудно обнаружить.* В своих версиях на начало 2008 года Xalan и ядро Altova XML игнорировали «плохие» идентификаторы, а MSXSL и Saxon не выдавали никаких полезных результатов.

[2.0] idref()

Для последовательности значений `ID` функция возвращает последовательность, содержащую все узлы, у которых значение `IDREF` совпадает с одним из заданных идентификаторов.

Синтаксис

```
node()* idref(xs:string*)
node()* idref(xs:string*, node())
```

Ввод

Последовательность значений `xs:string`, каждое из которых представляет значение `ID`. Если необязательный аргумент-узел присутствует, процессор ищет значение типа `IDREF` и `IDREFS` в документе, содержащем этот узел. Без аргумента `node()` процессор ищет совпадения в документе, содержащем контекстный узел.

Вывод

Последовательность узлов, каждый из которых содержит значение типа `IDREF` или `IDREFS`, совпадающее с одним из заданных значений `ID`. Узлы возвращаются в порядке документа, а все дубликаты узлов удаляются.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.5 «Functions and Operators that Generate Sequences».

Пример

В следующем примере используется список компонентов и деталей со встроенным описанием DTD. Мы перечислим все детали, указывая после каждой из них названия всех компонентов, использующих данную деталь. Функция `idref()` позволяет получить элемент `<component>` с заданным идентификатором; из него берется название компонента.

Данный пример является полной противоположностью примера для функции `id()`. Здесь мы перебираем элементы `<component>` и перечисляем все детали, которые в них используются.

Для демонстрации функции idref() используется документ XML со встроенным описанием DTD. (Это позволяет продемонстрировать работу функции без требования об обязательном использовании схемосовместимого процессора XSLT.) Фрагмент документа:

```
<?xml version="1.0"?>
<!-- parts-list1.xml -->
<!DOCTYPE parts-list [
<!ELEMENT parts-list       (component+, part+)>

<!ELEMENT component        (name, partref+)>
<!ATTLIST component        component-id ID #REQUIRED>

<!ELEMENT name             (#PCDATA)>

<!ELEMENT partref          EMPTY>
<!ATTLIST partref          refid IDREF #REQUIRED>

<!ELEMENT part             (name)>
<!ATTLIST part             part-id ID #REQUIRED
                           price CDATA #REQUIRED
                           source CDATA #REQUIRED>
]>

<parts-list>
  <component component-id="C28392-33-TT">
    <name>Turnip Twaddler</name>
    <partref refid="P81952-26-PK"/>
    <partref refid="P86679-52-SP"/>
    <partref refid="P81472-68-FD"/>
    <partref refid="P88107-39-GT"/>
  </component>
...
  <component component-id="C28772-63-OB">
    <name>Olive Bruiser</name>
    <partref refid="P80228-21-PT"/>
    <partref refid="P82387-85-PA"/>
  </component>

  <part part-id="P80228-21-PT" price="3.28" source="us">
    <name>Pitter</name>
  </part>
...
  <part part-id="P86994-25-RC" price="4.28" source="gb">
    <name>Ribbon Curler</name>
  </part>
</parts-list>
```

Документ содержит список компонентов; каждый компонент содержит атрибут component-id, элемент <name> и элементы <partref>. Каждый элемент <partref> содержит атрибут refid, ссылающийся на элемент <part>. Типы данных ID, IDREF и IDREFS (или xs:ID, xs:IDREF и xs:IDREFS в XML Schema) подчиняются следующим правилам:

- Любой атрибут типа ID (xs:ID) должен обладать уникальным значением, отличным от значений всех остальных атрибутов ID. Иначе говоря, в нашем документе атрибут component-id элемента <component> не может иметь такое же значение, как и атрибут part-id элемента <part>.
- Любой атрибут типа IDREF (xs:IDREF) должен иметь значение, соответствующее значению атрибута ID где-то в документе.

- Любой атрибут типа IDREFS (xs:IDREFS) должен содержать одно или несколько значений IDREF, разделенных пробелами.

Таблица стилей относительно проста; она перебирает все элементы <part> в документе и использует функцию idrefs() для поиска всех элементов <component>, использующих данную деталь:

```
<?xml version="1.0"?>
<!-- idref.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Here is a test of the idref() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:for-each select="/parts-list/part">
      <xsl:sort select="name"/>
      <xsl:text>&#xA;  </xsl:text>
      <xsl:value-of select="name"/>
      <xsl:text> (part #</xsl:text>
      <xsl:value-of select="@part-id"/>
      <xsl:text>) is used in these products:&#xA;    </xsl:text>
      <xsl:value-of select="idref(@part-id)/../../name"
        separator="&#xA;    "/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

При вызове функции idref() передается значение атрибута part-id для текущего элемента <part>. Функция idref() возвращает последовательность всех атрибутов типа IDREF, значения которых содержат текущее значение part-id.

Помните, что idref() возвращает последовательность самих атрибутов, а не их родителей; из-за этого выражение XPath для вывода названия каждой детали имеет вид idref(@part-id)/../../name. Функция idref() возвращает последовательность узлов атрибутов refid, в которых было найдено совпадение. Чтобы получить название каждой детали, мы сначала находим родителя refid (элемент <partref>), затем его родителя (элемент <component>) и, наконец, его потомка <name>. Результат выглядит так:

```
Here is a test of the idref() function:

  Feather Duster (part #P81472-68-FD) is used in these products:
    Turnip Twaddler
    Clam Teaser
    Cucumber Decorating Kit

  Grommet (part #P88107-39-GT) is used in these products:
    Turnip Twaddler
    Lemon Snubber

  Mucilage (part #P80499-43-MC) is used in these products:
    Prawn Goader
    Cucumber Decorating Kit
```

```
Paring Knife (part #P81952-26-PK) is used in these products:
  Turnip Twaddler
  Prawn Goader
  Cucumber Decorating Kit
  Lemon Snubber

Patter (part #P82387-85-PA) is used in these products:
  Olive Bruiser

Pitter (part #P80228-21-PT) is used in these products:
  Olive Bruiser

Ribbon Curler (part #P86994-25-RC) is used in these products:
  Prawn Goader
  Clam Teaser
  Cucumber Decorating Kit

Spanner (part #P86679-52-SP) is used in these products:
  Turnip Twaddler
  Clam Teaser
  Lemon Snubber
```

В исходном документе не используются атрибуты с типом данных `IDREFS` **(**`xs:IDREFS`**). При желании элемент** `<component>` **можно спроектировать следующим образом:**

```
<?xml version="1.0"?>
<!-- parts-list2.xml -->
<!DOCTYPE parts-list [
<!ELEMENT parts-list       (component+, part+)>

<!ELEMENT component        (name, partref)>
<!ATTLIST component        component-id ID #REQUIRED>

<!ELEMENT name             (#PCDATA)>

<!ELEMENT partref          EMPTY>
<!ATTLIST partref          refid IDREFS #REQUIRED>
<!ELEMENT part             (name) >
<!ATTLIST part             part-id ID #REQUIRED
                           price CDATA #REQUIRED
                           source CDATA #REQUIRED>
]>

<parts-list>
  <component component-id="C28392-33-TT">
    <name>Turnip Twaddler</name>
    <partref
      refid="P81952-26-PK P86679-52-SP P81472-68-FD P88107-39-GT"/>
  </component>
...
  <part part-id="P80228-21-PT" price="3.28" source="us">
    <name>Pitter</name>
  </part>
  <part part-id="P82387-85-PA" price="6.92" source="us">
    <name>Patter</name>
  </part>
...
</parts-list>
```

В этом варианте атрибут `refid` относится к типу данных `IDREFS` (`xs:IDREFS`). Это означает, что атрибут содержит список значений `IDREF`, разделенных пробелами. Из-за особенностей работы `idref()` таблицу стилей можно применить к документу с измененной структурой и получить тот же результат. Файл *parts-list2.xml* изменил структуру ссылок на идентификаторы деталей, но не изменил способ и место определения самих идентификаторов деталей; следовательно, функция `idref()` по-прежнему возвращает желаемый результат.

При помощи функции `idref()` мы получили информацию об использовании каждой детали (`<part>`) в системе учета.

Помните, что значения типов данных `ID`, `IDREF` и `IDREFS` должны представлять собой действительные имена XML. Идентификаторы, начинающиеся с цифры, могут привести к непредсказуемым результатам. Дополнительная информация приведена в конце описания функции `id()` ранее в этом приложении.

[2.0] implicit-timezone()

Возвращает часовой пояс по умолчанию, используемый при создании новых значений `xs:date`, `xs:dateTime` и `xs:time`.

Синтаксис

```
xs:dayTimeDuration implicit-timezone()
```

Ввод

Нет.

Вывод

Значение `xs:dayTimeDuration`, содержащее часовой пояс по умолчанию.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 16 «Context Functions».

Пример

Следующая таблица стилей возвращает информацию о часовом поясе по умолчанию в виде `xs:dayTimeDuration`:

```
<?xml version="1.0"?>
<!-- implicit-timezone.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Using the implicit-timezone() function:&#xA;&#xA;</xsl:text>
    <xsl:text>  The implicit timezone for the current context is: </xsl:text>
    <xsl:value-of select="implicit-timezone()"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Using the implicit-timezone() function:

  The implicit timezone for the current context is: -PT5H
```

Результаты для Бангалора и Лондона выглядят соответственно так:

```
  The implicit timezone for the current context is: PT5H30M
...
  The implicit timezone for the current context is: PT0S
```

[2.0] in-scope-prefixes()

Для заданного элемента возвращает последовательность всех префиксов пространств имен, находящихся в его области видимости.

Синтаксис

```
xs:string* in-scope-prefixes(element())
```

Ввод

Элемент.

Вывод

Последовательность значений `xs:string`, описывающих префиксы пространств имен в области видимости элемента.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 11.2 «Operators and Functions Related to QNames».

Пример

Для демонстрации работы функции `in-scope-prefixes()` мы снова воспользуемся документом с сонетом Шекспира:

```
<?xml version="1.0"?>
<!-- sonnet.xml -->
<sonnet type='Shakespearean'>
  <auth:author xmlns:auth="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </auth:author>
  <!-- Существует ли официальное название сонета?
       Иногда сонеты называются по первой строке. -->
  <title>Sonnet 130</title>
  <lines>
    <line>My mistress' eyes are nothing like the sun,</line>
    <line>Coral is far more red than her lips red.</line>
    <line>If snow be white, why then her breasts are dun,</line>
    <line>If hairs be wires, black wires grow on her head.</line>
    <line>I have seen roses damasked, red and white,</line>
```

```
    <line>But no such roses see I in her cheeks.</line>
    <line>And in some perfumes is there more delight</line>
    <line>Than in the breath that from my mistress reeks.</line>
    <line>I love to hear her speak, yet well I know</line>
    <line>That music hath a far more pleasing sound.</line>
    <line>I grant I never saw a goddess go,</line>
    <line>My mistress when she walks, treads on the ground.</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
  </lines>
</sonnet>
```

Следующая таблица стилей выводит все префиксы пространств имен, находящиеся в области видимости для каждого элемента в документе:

```
<?xml version="1.0"?>
<!-- in-scope-prefixes.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the in-scope-prefixes() function:</xsl:text>

    <xsl:for-each select="//*">
      <xsl:text>&#xA;    In-scope prefixes for &lt;</xsl:text>
      <xsl:value-of select="name()"/>
      <xsl:text>&gt;: </xsl:text>
      <xsl:value-of select="in-scope-prefixes(.)" separator=", "/>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
A test of the in-scope-prefixes() function:
    In-scope prefixes for <sonnet>: xml
    In-scope prefixes for <auth:author>: xml, auth
    In-scope prefixes for <last-name>: xml, auth
    In-scope prefixes for <first-name>: xml, auth
    In-scope prefixes for <nationality>: xml, auth
    In-scope prefixes for <year-of-birth>: xml, auth
    In-scope prefixes for <year-of-death>: xml, auth
    In-scope prefixes for <title>: xml
    In-scope prefixes for <lines>: xml
    In-scope prefixes for <line>: xml
    In-scope prefixes for <line>: xml
    In-scope prefixes for <line>: xml
    In-scope prefixes for <line>: xml
    In-scope prefixes for <line>: xml
    In-scope prefixes for <line>: xml
    In-scope prefixes for <line>: xml
    In-scope prefixes for <line>: xml
    In-scope prefixes for <line>: xml
    In-scope prefixes for <line>: xml
    In-scope prefixes for <line>: xml
    In-scope prefixes for <line>: xml
```

```
In-scope prefixes for <line>: xml
In-scope prefixes for <line>: xml
```

Префикс `xml` всегда находится в области видимости; по мере обработки исходного документа процессором XSLT префикс `auth` входит и выходит из области видимости.

[2.0] index-of()

Для заданных последовательности и условия поиска возвращает последовательность целых чисел, обозначающих позицию(-и) условия поиска в последовательности.

Синтаксис

```
xs:integer* index-of($sequenceParam as xs:anyAtomicType*,
                     $searchParam as xs:anyAtomicType)
xs:integer* index-of($sequenceParam as xs:anyAtomicType*,
                     $searchParam as xs:anyAtomicType,
                     $collation as xs:string)
```

Ввод

Последовательность атомарных значений и условие поиска (тоже атомарное значение). Необязательный третий аргумент задает URL-адрес схемы упорядочения, используемой при сравнении условия поиска со значениями в последовательности.

Вывод

Последовательность `xs:integer`, представляющая позицию(-и) условия поиска в исходной последовательности.

Если исходная последовательность является пустой последовательностью или ни одно значение в последовательности не совпадает с условием поиска, `index-of()` возвращает пустую последовательность. Кроме того, при сравнении условия поиска со значениями в последовательности процессор использует правила, определенные для оператора `eq` применительно к каждому типу данных. Если оператор `eq` не определен для конкретного типа данных, все значения этого типа считаются отличными от условия поиска.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.1 «General Functions and Operators on Sequences».

Пример

Короткая таблица стилей с использованием функции `index-of()`:

```
<?xml version="1.0"?>
<!-- index-of.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <xsl:variable name="now" as="xs:time"
      select="current-time()"/>
```

```
    <xsl:variable name="testSequence1" as="item()*">
      <xsl:sequence
        select="(3, 4, 5, 'blue', $now, current-date(), 8, 'blue',
                 'Strasse', 'Stra&#xDF;e')"/>
    </xsl:variable>

    <xsl:text>&#xA;Here's a test of the index-of() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  Our first test sequence ($testSequence1) </xsl:text>
    <xsl:text>is:&#xA;      </xsl:text>
    <xsl:value-of select="$testSequence1" separator="&#xA;      "/>
    <xsl:text>&#xA;&#xA;</xsl:text>

    <xsl:text>    $now = </xsl:text>
    <xsl:value-of select="$now"/>
    <xsl:text>&#xA;&#xA;</xsl:text>

    <xsl:text>    index-of($testSequence1, 3) = (</xsl:text>
    <xsl:value-of
      select="index-of($testSequence1, 3)" separator=", "/>
    <xsl:text>)&#xA;&#xA;</xsl:text>

    <xsl:text>    index-of($testSequence1, 'red') = (</xsl:text>
    <xsl:value-of
      select="index-of($testSequence1, 'red')" separator=", "/>
    <xsl:text>)&#xA;&#xA;</xsl:text>

    <xsl:text>    index-of($testSequence1, 'blue') = (</xsl:text>
    <xsl:value-of
      select="index-of($testSequence1, 'blue')" separator=", "/>
    <xsl:text>)&#xA;&#xA;</xsl:text>

    <xsl:text>    index-of($testSequence1, $now) = (</xsl:text>
    <xsl:value-of
      select="index-of($testSequence1, $now)" separator=", "/>
    <xsl:text>)</xsl:text>

    <xsl:text>&#xA;&#xA;    index-of($testSequence1, </xsl:text>
    <xsl:text>'stra&#xDF;e') = (</xsl:text>
    <xsl:value-of
      select="index-of($testSequence1, 'Strasse')"
      separator=", "/>
    <xsl:text>)&#xA;</xsl:text>

    <xsl:text>&#xA;    index-of($testSequence1, </xsl:text>
    <xsl:text>'stra&#xDF;e', [German collation]) = (</xsl:text>
    <xsl:value-of
      select="index-of($testSequence1, 'Stra&#xDF;e',
              concat('http://saxon.sf.net/collation?',
                     'class=com.oreilly.xslt.GermanCollation;'))"
      separator=", "/>
    <xsl:text>)&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here's a test of the index-of() function:

  Our first test sequence ($testSequence1) is:
```

```
    3
    4
    5
    blue
    15:04:18.841-05:00
    2006-11-23-05:00
    8
    blue
    Strasse
    Straße
$now = 15:04:18.841-05:00

index-of($testSequence1, 3) = (1)

index-of($testSequence1, 'red') = ()

index-of($testSequence1, 'blue') = (4, 8)

index-of($testSequence1, $now) = (5)

index-of($testSequence1, 'Strasse') = (9)

index-of($testSequence1, 'Strasse', [German collation]) = (9, 10)
```

Обратите внимание: мы сохраняем текущее время в переменной `$now`. С технической точки зрения делать это необязательно – функция `current-time()` стабильна; другими словами, при каждом вызове во время одного преобразования она возвращает один и тот же результат.

Обратите также внимание на использование нестандартного упорядочения при последнем вызове `index-of()`. В немецком языке *Strasse* и *Straße* – два варианта записи одного слова. Упорядочение по умолчанию игнорирует это обстоятельство, но оно отражено в нашем пользовательском упорядочении. (Чтобы листинг поместился на странице, мы объединили две половины URI-адреса схемы упорядочения Saxon при помощи функции `concat()`.) За дополнительной информацией обращайтесь к разделу «Функция document() и сортировка» главы 8.

[2.0] insert-before()

Позволяет создать новую последовательность посредством вставки объектов в существующую последовательность.

Синтаксис

```
item()* insert-before($target as item()*, $position as xs:integer,
                      $inserts as item()*)
```

Ввод

Последовательность, `xs:integer` (позиция) и вторая последовательность.

Вывод

Новая последовательность, полученная вставкой второй последовательности в первую последовательность перед объектом в заданной позиции.

В работе `insert-before()` используются вполне логичные, предсказуемые правила. Если первая последовательность пуста, вторая последовательность возвращается без изменений. Если вторая последовательность пуста, то первая последовательность

возвращается без изменений. Если позиция меньше 1, процессор вставляет вторую последовательность в начало первой. Наконец, если позиция больше количества объектов в первой последовательности, то вторая последовательность вставляется в конец первой.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.1 «General Functions and Operators on Sequences».

Пример

Таблица стилей с несколькими вариантами использования `insert-before()`:

```
<?xml version="1.0"?>
<!-- insert-before.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <xsl:variable name="longSequence" as="item()*">
      <xsl:sequence
        select="(3, 4, 5, current-date(), current-time(), 8, 'blue')"/>
    </xsl:variable>

    <xsl:variable name="shortSequence" as="item()*"
      select="(current-dateTime(), xs:yearMonthDuration('P3Y8M'))"/>

    <xsl:text>&#xA;Here's a test of the insert-before() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  Our first sequence ($longSequence) </xsl:text>
    <xsl:text>is:&#xA;      </xsl:text>
    <xsl:value-of select="$longSequence" separator="&#xA;      "/>

    <xsl:text>&#xA;&#xA;  Our second sequence ($shortSequence) </xsl:text>
    <xsl:text>is:&#xA;      </xsl:text>
    <xsl:value-of select="$shortSequence" separator="&#xA;      "/>

    <xsl:text>&#xA;&#xA;    Test 1. insert-before($longSequence, </xsl:text>
    <xsl:text>3, (23)) </xsl:text>
    <xsl:text>= &#xA;      </xsl:text>
    <xsl:value-of select="insert-before($longSequence, 3, (23))"
      separator="&#xA;      "/>

    <xsl:text>&#xA;&#xA;    Test 2. insert-before($longSequence, </xsl:text>
    <xsl:text>4, $shortSequence) </xsl:text>
    <xsl:text>= &#xA;      </xsl:text>
    <xsl:value-of
      select="insert-before($longSequence, 4, $shortSequence)"
      separator="&#xA;      "/>

    <xsl:text>&#xA;&#xA;    Test 3. insert-before($longSequence, </xsl:text>
    <xsl:text>4, $shortSequence) </xsl:text>
    <xsl:text>= &#xA;      </xsl:text>
    <xsl:value-of
      select="insert-before($longSequence, 4, $shortSequence)"
```

```
      separator="&#xA;      "/>

    <xsl:text>&#xA;&#xA;    Test 4. insert-before($longSequence, 0, </xsl:text>
    <xsl:text>$shortSequence) </xsl:text>
    <xsl:text>= &#xA;      </xsl:text>
     <xsl:value-of select="insert-before($longSequence, 0, $shortSequence)"
      separator="&#xA;      "/>

    <xsl:text>&#xA;&#xA;    Test 5. insert-before($longSequence, 42, </xsl:text>
    <xsl:text>$shortSequence) </xsl:text>
    <xsl:text>= &#xA;      </xsl:text>
    <xsl:value-of select="insert-before($longSequence, 42, $shortSequence)"
      separator="&#xA;      "/>

    <xsl:text>&#xA;&#xA;    Creating a new variable based on </xsl:text>
    <xsl:text>&#xA;        </xsl:text>
    <xsl:text>insert-before($longSequence, 4, $shortSequence): </xsl:text>
    <xsl:text>&#xA;      </xsl:text>
    <xsl:text>updatedSequence = insert-before($longSequence, 4, </xsl:text>
    <xsl:text>$shortSequence) : &#xA;      </xsl:text>
    <xsl:variable name="updatedSequence" as="item()*"
      select="insert-before($longSequence, 4, $shortSequence)"/>
    <xsl:text>&#xA;    Test 6. $updatedSequence = &#xA;      </xsl:text>
    <xsl:value-of select="$updatedSequence" separator="&#xA;      "/>

  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here's a test of the insert-before() function:

  Our first sequence ($longSequence) is:
      3
      4
      5
      2006-07-25-04:00
      04:24:21.216-04:00
      8
      blue

  Our second sequence ($shortSequence) is:
      2006-07-25T04:24:21.216-04:00
      P3Y8M

    Test 1. insert-before($longSequence, 3, (23)) =
      3
      4
      23
      5
      2006-07-25-04:00
      04:31:23.383-04:00
      8
      blue

    Test 2. insert-before($longSequence, 4, $shortSequence) =
      3
      4
      5
      2006-07-25T04:24:21.216-04:00
```

```
  P3Y8M
  2006-07-25-04:00
  04:24:21.216-04:00
  8
  blue

Test 3. insert-before($longSequence, 4, $shortSequence) =
  3
  4
  5
  2006-07-25T04:24:21.216-04:00
  P3Y8M
  2006-07-25-04:00
  04:24:21.216-04:00
  8
  blue

Test 4. insert-before($longSequence, 0, $shortSequence) =
  2006-07-25T04:24:21.216-04:00
  P3Y8M
  3
  4
  5
  2006-07-25-04:00
  04:24:21.216-04:00
  8
  blue

Test 5. insert-before($longSequence, 42, $shortSequence) =
  3
  4
  5
  2006-07-25-04:00
  04:24:21.216-04:00
  8
  blue
  2006-07-25T04:24:21.216-04:00
  P3Y8M

Creating a new variable based on
    insert-before($longSequence, 4, $shortSequence):
  updatedSequence = insert-before($longSequence, 4, $shortSequence) :

Test 6. $updatedSequence =
  3
  4
  5
  2006-07-25T04:24:21.216-04:00
  P3Y8M
  2006-07-25-04:00
  04:24:21.216-04:00
  8
  blue
```

Таблица стилей начинается с вывода двух последовательностей. В первом тесте создается и вставляется жестко закодированная последовательность (синглет со значением 23). В тесте 2 функция вставляет короткую последовательность в более длинную перед позицией 4 и возвращает полученный результат. Тест 3 показывает, что вызов

`insert-before()` не изменяет исходной последовательности; функция просто возвращает новую последовательность со вставленными значениями.

Тесты 4 и 5 показывают, что происходит при выходе позиции за пределы диапазона допустимых значений. При задании позиции `0` вторая последовательность вставляется в начало первой, а если позиция равна `42`, вторая последовательность вставляется в конец первой. Наконец, тест 6 сохраняет новую последовательность в переменной.

[2.0] iri-to-uri()

Преобразует строку, содержащую IRI (Internationalized Resource Identifier), в URI-адрес. В синтаксисе URI может использоваться подмножество из примерно 60 символов ASCII, чего определенно недостаточно для большинства мировых языков. Синтаксис IRI решает эту проблему; данная функция преобразует синтаксис IRI в синтаксис URI.

Синтаксис

```
xs:string iri-to-uri(xs:string?)
```

Ввод

Значение `xs:string`, содержащее IRI.

Вывод

Значение `xs:string`, в котором необходимые символы закодированы для использования в URI. (За подробностями обращайтесь к документу RFC 3987, Internationalized Resource Identifiers [IRIs], по адресу *http://www.ietf.org/rfc/rfc3987.txt*. Полная информация о синтаксисе URI содержится в документе RFC 3986, Uniform Resource Identifiers [URI]: Generic Syntax, по адресу *http://www.ietf.org/rfc/rfc3986.txt*.)

Функция *не* кодирует знак процента (`%`). Чтобы использовать эту функцию, вам придется закодировать все проценты самостоятельно (задача решается вызовом `replace-string($string, '%', '%25')`) перед вызовом `iri-to-uri()`.

Если значение аргумента представляет собой пустую последовательность, функция возвращает строку нулевой длины.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.4 «Functions on String Values».

Пример

Два примера из спецификации:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- iri-to-uri.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the iri-to-uri() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  iri-to-uri('http://www.example.com/</xsl:text>
```

```
    <xsl:text>00/Weather/CA/Los Angeles#ocean') = &#xA;    </xsl:text>
    <xsl:value-of
      select="iri-to-uri('http://www.example.com/00/Weather/CA/Los Angeles#ocean')"/>

    <xsl:text>&#xA;&#xA;  iri-to-uri('http://www.example.com/</xsl:text>
    <xsl:text>~b&#xE9;b&#xE9;') = &#xA;    </xsl:text>
    <xsl:value-of
      select="iri-to-uri('http://www.example.com/~b&#xE9;b&#xE9;')"/>

    <!-- Символ &#x5BB6; в японском языке означает "дом" -->
    <xsl:text>&#xA;&#xA;  iri-to-uri('http://www.example.co.jp/</xsl:text>
    <xsl:text>&#x5BB6;') = &#xA;    </xsl:text>
    <xsl:value-of
      select="iri-to-uri('http://www.example.co.jp/&#x5BB6;')"/>
  </xsl:template>

</xsl:stylesheet>
```

Как видно из рис. С.4, IRI преобразуются в URI-адреса.

```
Tests of the iri-to-uri() function:

iri-to-uri('http://www.example.com/00/Weather/CA/Los Angeles#ocean') =
  http://www.example.com/00/Weather/CA/Los%20Angeles#ocean

iri-to-uri('http://www.example.com/~bébé') =
  http://www.example.com/~b%C3%A9b%C3%A9
iri-to-uri('http://www.example.co.jp/家') =
  http://www.example.co.jp/%E5%AE%B6
```

***Рис. С.4.** Пример с кодированием символов, недопустимых в URI*

В первом примере пробел кодируется в виде `%20`; во втором символ *é* кодируется двухбайтовой последовательностью UTF-8 `%C3%A9`.

Японский иероглиф из последнего примера (家) кодируется трехбайтовой последовательностью UTF-8 `%E5%AE%B6`. (По-моему, этот иероглиф означает «дом», но я не уверен на 100%.)

См. также

Описания функций ***[2.0]*** `encode-for-uri()` и ***[2.0]*** `escape-html-uri()`.

key()

Ссылается на отношение, определяемое элементом `<xsl:key>`. На концептуальном уровне функция `key()` работает аналогично функции `id()`, однако механизм ключей более гибок, чем механизм идентификаторов.

Синтаксис

```
[1.0] node-set key(string, object)
[2.0] node()* key(xs:string, xs:anyAtomicType*)
[2.0] node()* key(xs:string, xs:anyAtomicType*, node())
```

Ввод

[1.0] Имя ключа (определяемое элементом `<xsl:key>`) и объект. Если объект представляет собой набор узлов, функция `key()` применяется к строковому значению каждого узла в наборе узлов и возвращает набор узлов, полученный в результате всех вызовов функции `key()`. Если объект относится к другому типу, он преобразуется в строку, как при вызове функции `string()`.

[2.0] Имя ключа (`xs:string`) и последовательность условий поиска. Необязательный третий аргумент ограничивает поиск теми узлами, у которых заданный узел принадлежит к оси `ancestor-or-self`. Третий аргумент позволяет ограничить результаты поиска конкретным набором узлов.

Вывод

[1.0] Набор узлов, содержащий узлы документа, включающего контекстный узел, у которых значение указанного ключа совпадает с условием(-ями) поиска. Иначе говоря, если наша таблица стилей содержит элемент `<xsl:key>`, определяющий ключ с именем `postalcodes` на основании дочернего элемента `<postalcode>` всех элементов `<address>` в текущем документе, вызов `key(postalcodes, '34829')` вернет набор узлов со всеми элементами `<address>`, у которых элемент `<postalcode>` содержит значение `34829`.

[2.0] Последовательность узлов, каждый из которых совпадает с одним из значений в последовательности условий поиска. Способ совпадения узлов с условиями поиска определяется заданным ключом.

[2.0] Помните, что любые нетипизованные значения в последовательности условий поиска будут сравниваться в строковом виде. Если используемый ключ определяет значения `xs:date` в качестве индекса, строковые значения никогда не совпадут, и `key()` всегда будет возвращать пустую последовательность. В этом случае результаты можно получить только при использовании условий поиска типа `xs:date`.

Где определяется

[1.0] XSLT, раздел 12.2 «Keys».

[2.0] XSLT, раздел 16.3 «Keys».

Пример

Для демонстрации возможностей функции `key()` мы используем следующий документ:

```
<?xml version="1.0"?>
<!-- simplified-names.xml -->
<addressbook>
  <address>
    <first-name>Chester Hasbrouck</first-name>
    <last-name>Frisby</last-name>
    <city>Sheboygan</city>
    <state>WI</state>
  </address>
  <address>
    <first-name>Mary</first-name>
    <last-name>Backstayge</last-name>
    <city>Skunk Haven</city>
```

```
    <state>MA</state>
  </address>
  <address>
    <first-name>Natalie</first-name>
    <last-name>Attired</last-name>
    <city>Winter Harbor</city>
    <state>ME</state>
  </address>
  <address>
    <first-name>Harry</first-name>
    <last-name>Backstayge</last-name>
    <city>Skunk Haven</city>
    <state>MA</state>
  </address>
  <address>
    <first-name>Mary</first-name>
    <last-name>McGoon</last-name>
    <city>Boylston</city>
    <state>VA</state>
  </address>
  <address>
    <first-name>Amanda</first-name>
    <last-name>Reckonwith</last-name>
    <city>Lynn</city>
    <state>MA</state>
  </address>
</addressbook>
```

А вот как выглядит таблица стилей, которая будет использоваться для обработки этого документа. Обратите внимание на определение ключа для индексирования адресов клиентов по штату:

```
<?xml version="1.0"?>
<!-- key-function.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:param name="searchState"/>

  <xsl:key name="customersByState" match="address" use="state"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the key() function:&#xA;</xsl:text>
    <xsl:text>&#xA;  All customers in </xsl:text>
    <xsl:value-of select="$searchState"/>
    <xsl:text>&#xA;&#xA;    </xsl:text>
    <xsl:for-each select="key('customersByState', $searchState)">
      <xsl:value-of select="first-name"/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="last-name"/>
      <xsl:text>, </xsl:text>
      <xsl:value-of select="city"/>
      <xsl:text>&#xA;    </xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

При выполнении таблицы стилей с параметром `$searchState`, равным `MA`, будет получен следующий результат:

```
A test of the key() function:

  All customers in MA

    Mary Backstayge, Skunk Haven
    Harry Backstayge, Skunk Haven
    Amanda Reckonwith, Lynn
```

При условии поиска `WI` обнаруживается только один клиент:

```
A test of the key() function:

  All customers in WI

    Chester Hasbrouck Frisby, Sheboygan
```

lang()

Определяет, совпадает ли код языка заданной строки с языком контекстного узла, определяемым атрибутом `xml:lang` (или является ли его подъязыком).

Синтаксис

```
[1.0] boolean lang(string)
[2.0] xs:boolean lang(xs:string?)
[2.0] xs:boolean lang(xs:string?, node()?)
```

Ввод

Строка, представляющая код языка. Если в контекстном узле задан язык `xml:lang="en-us"`, вызов функции `lang()` с любым из значений `en`, `EN` и `en-us` возвращает логическое значение `true`, а вызов `lang()` со значением `en-gb` возвращает логическое значение `false`.

Вывод

Если код языка строки аргумента совпадает с языком контекстного узла (или является его подъязыком), `lang()` возвращает логическое значение `true`. Если контекстный узел не имеет атрибута `xml:lang`, вместо него используется значение атрибута `xml:lang` ближайшего предка. Если такой атрибут тоже отсутствует, функция `lang()` возвращает логическое значение `false`. При сравнении кода языка контекстного узла с кодом языка строки аргумента функция `lang()` игнорирует регистр символов.

Где определяется

[1.0] XPath, раздел 4.3 «Boolean Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 14 «Functions and Operators on Nodes».

Пример

Документ XML с кодами языков:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
```

```
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Таблица стилей, демонстрирующая использование функции `lang()`**:**

```
<?xml version="1.0"?>
<!-- lang.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the lang() function:&#xA;&#xA;</xsl:text>
    <xsl:for-each select="list/listitem">
      <xsl:choose>
        <xsl:when test="lang('EN')">
          <xsl:text>  Here's an English-language album: </xsl:text>
        </xsl:when>
        <xsl:otherwise>
          <xsl:text>  -------> Here's some World music: </xsl:text>
        </xsl:otherwise>
      </xsl:choose>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Наконец, результат преобразования:

```
A test of the lang() function:

  Here's an English-language album: The Sacred Art of Dub
  Here's an English-language album: Only the Poor Man Feel It
  Here's an English-language album: Excitable Boy
  -------> Here's some World music: Aki Special
  Here's an English-language album: Combat Rock
  -------> Here's some World music: Talking Timbuktu
  -------> Here's some World music: The Birth of the Cool
```

Обратите внимание: условие поиска `EN` **совпадает как с** `en` **(язык по умолчанию), так и с** `en-gb` **(британский английский).**

last()

Возвращает позицию последнего узла в текущем контексте. При помощи этой функции можно, например, определить шаблон последнего вхождения заданного элемента или проверить, является ли заданный узел последним в наборе узлов, к которому он принадлежит.

Синтаксис

```
[1.0] number last()
[2.0] xs:integer last()
```

Ввод

Нет.

Вывод

Число, равное количеству узлов в текущем контексте. Например, если текущий контекст содержит 12 узлов <li>, last() вернет 12.

Где определяется

[1.0] XPath, раздел 4.1 «Node Set Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 16 «Context Functions».

Пример

В следующем примере функция last() обеспечивает особую обработку последнего элемента списка. Мы используем следующий документ XML:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Следующая таблица стилей обрабатывает последний элемент <listitem> особым образом:

```
<?xml version="1.0"?>
<!-- last.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the last() function:&#xA;</xsl:text>
    <xsl:for-each select="list/listitem">
      <xsl:text>&#xA;  </xsl:text>
      <xsl:if test="position()=last()">
        <xsl:text>&#xA;  LAST, but not least:  </xsl:text>
      </xsl:if>
      <xsl:value-of select="."/>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

При преобразовании документа XML по этой таблице стилей будет получен следующий результат:

```
A test of the last() function:

  The Sacred Art of Dub
  Only the Poor Man Feel It
  Excitable Boy
  Aki Special
  Combat Rock
  Talking Timbuktu

  LAST, but not least:  The Birth of the Cool
```

local-name()

Возвращает локальную часть первого узла в наборе узлов, переданном в аргументе.

Синтаксис

```
[1.0] string local-name(node-set?)
[2.0] xs:string local-name()
[2.0] xs:string local-name(node()?)
```

Ввод

Набор узлов. Если входной набор узлов пуст, функция возвращает пустую строку. Если набор узлов не указан, функция использует набор узлов, состоящий только из контекстного узла.

Вывод

Строка, соответствующая локальному имени первого элемента в наборе узлов, переданном в аргументе. Если набор узлов пуст, функция `local-name()` возвращает пустую строку.

[2.0] В XSLT 2.0 передача аргумента, содержащего более одного узла, приводит к фатальной ошибке.

Где определяется

[1.0] XPath, раздел 4.1 «Node Set Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 14 «Functions and Operators on Nodes».

Пример

Для демонстрации возможностей функции `local-name()` используется документ XML с сонетом Шекспира:

```
<?xml version="1.0"?>
<!-- sonnet.xml -->
<sonnet type='Shakespearean'>
  <auth:author xmlns:auth="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
```

```
    <year-of-death>1616</year-of-death>
  </auth:author>
  <!-- Существует ли официальное название сонета?
       Иногда сонеты называются по первой строке. -->
  <title>Sonnet 130</title>
  <lines>
    <line>My mistress' eyes are nothing like the sun,</line>
    <line>Coral is far more red than her lips red.</line>
    <line>If snow be white, why then her breasts are dun,</line>
    <line>If hairs be wires, black wires grow on her head.</line>
    <line>I have seen roses damasked, red and white,</line>
    <line>But no such roses see I in her cheeks.</line>
    <line>And in some perfumes is there more delight</line>
    <line>Than in the breath that from my mistress reeks.</line>
    <line>I love to hear her speak, yet well I know</line>
    <line>That music hath a far more pleasing sound.</line>
    <line>I grant I never saw a goddess go,</line>
    <line>My mistress when she walks, treads on the ground.</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
  </lines>
</sonnet>
```

Короткая таблица стилей, используемая для преобразования документа:

```
<?xml version="1.0"?>
<!-- local-name.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the local-name() function:&#xA;</xsl:text>

    <xsl:for-each select="//*">
      <xsl:text>&#xA;    The local name for &lt;</xsl:text>
      <xsl:value-of select="name()"/>
      <xsl:text>&gt;: </xsl:text>
      <xsl:value-of select="local-name(.)"/>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
A test of the local-name() function:

    The local name for <sonnet>: sonnet
    The local name for <auth:author>: author
    The local name for <last-name>: last-name
    The local name for <first-name>: first-name
    The local name for <nationality>: nationality
    The local name for <year-of-birth>: year-of-birth
    The local name for <year-of-death>: year-of-death
    The local name for <title>: title
    The local name for <lines>: lines
    The local name for <line>: line
    The local name for <line>: line
```

```
The local name for <line>: line
The local name for <line>: line
The local name for <line>: line
The local name for <line>: line
The local name for <line>: line
The local name for <line>: line
The local name for <line>: line
The local name for <line>: line
The local name for <line>: line
The local name for <line>: line
The local name for <line>: line
The local name for <line>: line
```

[2.0] local-name-from-QName()

Получает значение `xs:QName` и возвращает локальное имя.

Синтаксис

```
xs:NCName? local-name-from-QName(xs:QName?)
```

Ввод

Значение `xs:QName`.

Вывод

Локальное имя из заданного полного имени. Данные возвращаются в формате `xs:NCName` (имя «без двоеточия»). Если аргумент является пустой последовательностью, `local-name-from-QName()` возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 11.2 «Operators and Functions Related to QNames».

Пример

Короткая таблица стилей, использующая функцию `local-name-from-QName()`:

```
<?xml version="1.0"?>
<!-- local-name-from-qname.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the local-name-from-QName() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:variable name="testQName1" as="xs:QName"
      select="QName('http://www.authors.com', 'auth:author')"/>

    <xsl:text>&#xA;  QName('http://www.authors.com', </xsl:text>
    <xsl:text>'auth:author') = </xsl:text>
    <xsl:value-of select="$testQName1"/>

    <xsl:text>&#xA;    The local name associated with </xsl:text>
```

```
    <xsl:text>this QName: "</xsl:text>
    <xsl:value-of select="local-name-from-QName($testQName1)"/>
    <xsl:text>"</xsl:text>

    <xsl:variable name="testQName2" as="xs:QName"
      select="QName('', 'sonnet')"/>

    <xsl:text>&#xA;&#xA;  QName('', 'sonnet') = </xsl:text>
    <xsl:value-of select="$testQName2"/>

    <xsl:text>&#xA;    The local name associated with </xsl:text>
    <xsl:text>this QName: "</xsl:text>
    <xsl:value-of select="local-name-from-QName($testQName2)"/>
    <xsl:text>"</xsl:text>

  </xsl:template>

</xsl:stylesheet>
```

Результат вполне тривиален:

```
Tests of the local-name-from-QName() function:

  QName('http://www.authors.com', 'auth:author') = auth:author
    The local name associated with this QName: "author"

  QName('', 'sonnet') = sonnet
    The local name associated with this QName: "sonnet"
```

[2.0] lower-case()

Преобразует строку к нижнему регистру.

Синтаксис

```
xs:string lower-case(xs:string)
```

Ввод

Значение `xs:string`.

Вывод

Значение `xs:string`, в котором все прописные буквы оригинала преобразованы в строчные. Все символы, изначально записанные в нижнем регистре, а также символы, не имеющие отдельной разновидности в нижнем регистре, возвращаются в неизменном виде. Если значение аргумента является пустой строкой, функция возвращает строку нулевой длины.

Из-за диакритических знаков и других особенностей национальных языков при приведении к нижнему регистру длина строки может измениться. Следует также учитывать, что в некоторых языках функции `lower-case()` и `upper-case()` не являются взаимно противоположными. Все правила преобразования регистра определяются в стандарте Юникода и должны соблюдаться процессорами XSLT.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.4 «Functions on String Values».

Пример

Следующая таблица стилей демонстрирует использование функции `lower-case()`**. Обратите внимание на элемент** `<xsl:output method="xml" encoding="UTF-8"/>`**, обеспечивающий правильную обработку кодировки символов:**

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- lower-case.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" encoding="UTF-8" indent="yes"/>

  <xsl:template match="/">
    <testcase>
      <heading>Tests of the lower-case() function:</heading>
      <test>
        <label>lower-case('Lily') = </label>
        <result><xsl:value-of select="lower-case('Lily')"/></result>
      </test>
      <test>
        <label>lower-case('LILY') = </label>
        <result><xsl:value-of select="lower-case('LILY')"/></result>
      </test>
      <test>
        <label>lower-case('lily') = </label>
        <result><xsl:value-of select="lower-case('lily')"/></result>
      </test>
      <test>
        <label>lower-case('JALAPECO') = </label>
        <result><xsl:value-of select="lower-case('JALAPECO')"/></result>
      </test>
    </testcase>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
<?xml version="1.0" encoding="UTF-8"?>
<testcase>
   <heading>Tests of the lower-case() function:</heading>
   <test>
      <label>lower-case('Lily') = </label>
      <result>lily</result>
   </test>
   <test>
      <label>lower-case('LILY') = </label>
      <result>lily</result>
   </test>
   <test>
      <label>lower-case('lily') = </label>
      <result>lily</result>
   </test>
   <test>
      <label>lowercase('JALAPECO') = </label>
      <result>jalapeco</result>
   </test>
</testcase>
```

[2.0] matches()

Проверяет, совпадает ли заданное регулярное выражение в заданной строке.

Синтаксис

```
xs:boolean matches($input as xs:string?, $pattern as xs:string)
xs:boolean matches($input as xs:string?, $pattern as xs:string,
                   $flags as xs:string)
```

Ввод

Входная строка и регулярное выражение. Функция `matches()` также может получать необязательную строку с флагами, управляющими обработкой регулярного выражения.

Вывод

`true`, если регулярное выражение совпадает в строке; `false` в противном случае.

Регулярные выражения представляют собой мощный механизм обработки текстов с широкими возможностями. Несколько замечаний по поводу их работы:

- Если аргумент `$input` является пустой последовательностью, он интерпретируется как строка нулевой длины. Даже в такой ситуации функция `matches()` может вернуть `true`. Например, значение `matches((), '.?')` равно `true`, как и значение `matches('', '.?')`.
- При поиске совпадений для регулярных выражений не используются схемы упорядочения; сравниваются коды символов в Юникоде. Ситуации, в которых разные символы считаются одинаковыми в национальных языках, не учитываются.
- Если только в регулярном выражении не используются якорные метасимволы ^ (начало строки) и $ (конец строки), совпадение считается обнаруженным в том случае, если шаблон совпадает с любой частью строки.
- В отличие от регулярных выражений, используемых в элементе `<xsl:analyze-string>`, фигурные скобки ({ и }) не удваиваются. (Фигурные скобки в атрибуте `regex` элемента `<xsl:analyze-string>` должны удваиваться, чтобы они не интерпретировались как шаблоны значений атрибутов.)
- Параметр `$flags` управляет обработкой регулярного выражения. Поддерживаются четыре флага:

  `s` Регулярное выражение обрабатывается в режиме «совпадения точки со всеми символами». При наличии этого флага метасимвол «точка» (`.`) совпадает с любым символом. При стандартной обработке (без флага `s`) оператор «точка» совпадает с любым символом, *кроме* символа новой строки (`
`). Этот флаг особенно полезен при обработке строк, содержащих внутренние символы новой строки.

  `m` Регулярное выражение обрабатывается в многострочном режиме. В режиме по умолчанию метасимвол ^ совпадает с началом всей строки, а метасимвол $ совпадает с концом каждой строки. В многострочном режиме ^ совпадает с началом, а $ – с концом любой логической строки внутри «физической» строки.

  `i` Регулярное выражение обрабатывается без учета регистра символов. Регулярное выражение `"a"` совпадает как с `"a"`, так и с `"A"`.

Учтите, что Юникод может создать массу проблем с этим флагом. Например, в спецификации XQuery 1.0 and XPath 2.0 Functions and Operators приводится пример с обозначением в Юникоде градусов по Кельвину (`K`), соответствующим букве `"K"`. Комбинация `regex="k"` и `flags="i"` совпадает как со знаком градусов по Кельвину, так и с буквами `"k"` (`k`) и `"K"` (`K`).

Другие символы Юникода не преобразуются в буквы. Например, символ римской цифры `I` (`ࡰ`) внешне похож на букву `I`, но не преобразуется в нее.

`x` Перед поиском совпадения из регулярного выражения удаляются все символы пропусков (`	`, `
`, `
` и ` `). Другими словами, с флагом `x` регулярные выражения `"John Smith"` и `"JohnSmith"` эквивалентны. Этот флаг особенно удобен при разбиении длинного регулярного выражения на несколько строк, чтобы его было легче читать.

Флаги могут комбинироваться в любом порядке. Наборы параметров `'xis'` и `'six'` работают абсолютно одинаково.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.6 «String Functions that Use Pattern Matching».

Пример

Таблица стилей с использованием функции `matches()`:

```
<?xml version="1.0"?>
<!-- matches.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <xsl:variable name="string1"
      select="concat('Now is the time for all good men&#xA;',
              'and women &#xA;',
              'to aid the party.')"/>

    <xsl:text>&#xA;Here's a test of the matches() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  $string1 = &#xA;</xsl:text>
    <xsl:value-of select="$string1"/>

    <xsl:text>&#xA;&#xA;  Test 1. matches($string1, </xsl:text>
    <xsl:text>'Now.*men') = </xsl:text>
    <xsl:value-of select="matches($string1, 'Now.*men')"/>

    <xsl:text>&#xA;&#xA;  Test 2. matches($string1, </xsl:text>
    <xsl:text>'Now.*men$') = </xsl:text>
    <xsl:value-of select="matches($string1, 'Now.*men$')"/>

    <xsl:text>&#xA;&#xA;  Test 3. matches($string1, </xsl:text>
    <xsl:text>'Now.*men$', 'm') = </xsl:text>
    <xsl:value-of select="matches($string1, 'Now.*men$', 'm')"/>

    <xsl:text>&#xA;&#xA;  Test 4. matches($string1, </xsl:text>
    <xsl:text>'women $', 'm') = </xsl:text>
```

```
    <xsl:value-of select="matches($string1, 'women $', 'm')"/>

    <xsl:text>&#xA;&#xA;  Test 5. matches($string1, </xsl:text>
    <xsl:text>'party.$') = </xsl:text>
    <xsl:value-of select="matches($string1, 'party.$')"/>

    <xsl:text>&#xA;&#xA;  Test 6. matches($string1, </xsl:text>
    <xsl:text>'^and' 'm') = </xsl:text>
    <xsl:value-of select="matches($string1, '^and', 'm')"/>

    <xsl:text>&#xA;&#xA;  Test 7. matches($string1, </xsl:text>
    <xsl:text>'women .?to' 's') = </xsl:text>
    <xsl:value-of select="matches($string1, 'women .?to', 's')"/>

  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here's a test of the matches() function:

  $string1 =
Now is the time for all good men
and women
to aid the party.

  Test 1. matches($string1, 'Now.*men') = true

  Test 2. matches($string1, 'Now.*men$') = false

  Test 3. matches($string1, 'Now.*men$', 'm') = true

  Test 4. matches($string1, 'women $', 'm') = true

  Test 5. matches($string1, 'party.$') = true

  Test 6. matches($string1, '^and' 'm') = true

  Test 7. matches($string1, 'women .?to' 's') = true
```

Строка из примера содержит два разрыва, чтобы мы могли протестировать работу метасимволов ^ и $ и действие флагов. Регулярное выражение в тесте 1 состоит из символов `'Now'`, за которыми следует нуль или более символов, сопровождаемых символами `'men'`. Тест 2 указывает, что символы `'men'` должны быть последними тремя символами в строке; в нашем случае это очевидно не так. В тесте 3 мы указываем, что выражение должно обрабатываться в многострочном режиме (с флагом `'m'`), чтобы подстрока `'men
'` совпала с выражением `'men$'`.

В тесте 4 мы ищем символы `'women '` в конце строки. Так как при поиске используется многострочный режим, совпадение находится успешно. Обратите внимание на необходимость включения пробела в конец строки; для выражения `'women$'` совпадения не будет. В тесте 5 строка должна завершаться символами `'party.'`. Многострочный режим здесь не действует, но это нормально; строка действительно завершается символами `'party.'`. Тест 6 ищет символы `'and'` в начале строки; так как в данном случае действует многострочный режим, поиск проходит успешно.

Наконец, мы ищем символы `'women '`, за которыми следует один любой необязательный символ, сопровождаемый символами `'to'`. В данном случае действует режим совпадения точки со всеми символами (мы задаем флаг `'s'`), так что совпадение находится успешно. Необязательным символом здесь оказывается символ новой строки, который совпадает с точкой при действующем флаге `'s'`.

См. также

В приложении E приведено подробное описание работы регулярных выражений в XPath 2.0. Обращайтесь также к определениям следующих элементов и функций: *[2.0]* <xsl:analyze-string>, *[2.0]* <xsl:matching-substring>, *[2.0]* <xsl:non-matching-substring>, *[2.0]* regex-group(), *[2.0]* replace() и *[2.0]* tokenize().

[2.0] max()

Возвращает наибольшее значение из заданной последовательности.

Синтаксис

```
xs:anyAtomicType max(xs:anyAtomicType*)
xs:anyAtomicType max(xs:anyAtomicType*, $collation as xs:string)
```

Ввод

Последовательность значений.

Вывод

Наибольшее значение из заданной последовательности. Функция max() работает со следующими типами значений: с числами (xs:integer, xs:double, xs:decimal и xs:float), с xs:string, xs:boolean, xs:date, xs:dateTime, xs:time, xs:yearMonthDuration и xs:dayTimeDuration (но *не* с xs:duration).

Для последовательности числовых значений процессор XSLT возвращает наибольшее число, при необходимости преобразуя типы данных. Для последовательности временных промежутков процессор XSLT возвращает промежуток наибольшей продолжительности. Для последовательности строк процессор XSLT возвращает строку, стоящую на последнем месте в порядке сортировки.

Функция max() предполагает, что ей передается последовательность, содержащая осмысленные данные; в противном случае процессор XSLT выдает ошибку. Например, при попытке получения наибольшего значения в последовательности (xs:dayTimeDuration('P3DT8H17M'), 38, 'football') произойдет ошибка – впрочем, как и следовало ожидать.

Несколько замечаний по поводу работы max():

- Для поиска самого длинного временного промежутка в последовательности либо все значения должны относиться к типу xs:dayTimeDuration, либо все значения должны относиться к типу xs:yearMonthDuration. Смешивать эти две разновидности промежутков нельзя; при нарушении этого требования процессор XSLT выдает ошибку.
- Если в последовательности присутствуют значения типа xs:untypedAtomic, процессор XSLT пытается преобразовать их к типу xs:double. Если объект не удается преобразовать к xs:double, процессор XSLT выдает ошибку.
- При сравнении значений xs:string используется схема упорядочения по умолчанию, если только вы явно не зададите другую схему.
- Наконец, при передаче пустой последовательности функции max() она возвращает пустую последовательность. Хотя функции в этом случае не передаются никакие полезные данные, процессор XSLT не выдает ошибку.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.4 «Aggregate Functions».

Пример

Следующая таблица стилей находит наибольшее значение в последовательностях нескольких разных типов:

```
<?xml version="1.0"?>
<!-- max.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="seq1" select="(3, 5, 18)"/>
    <xsl:variable name="seq2" select="(3, 5, 48.273, 2.9e3)"/>

    <xsl:variable name="value1" as="xs:integer" select="42"/>
    <xsl:variable name="value2" as="xs:double" select="2718.28E-3"/>
    <xsl:variable name="value3" as="xs:float" select="98.6"/>
    <xsl:variable name="value4" as="xs:decimal" select="2.54"/>
    <xsl:variable name="seq3"
      select="($value1, $value2, $value3, $value4)"/>

    <xsl:variable name="seq4"
      select="(xs:yearMonthDuration('P3Y8M'),
               xs:yearMonthDuration('P4Y2M'),
               xs:yearMonthDuration('P6Y4M'))"/>
    <xsl:variable name="seq5"
      select="(xs:dayTimeDuration('P2DT4H23M12.2S'),
               xs:dayTimeDuration('P3DT8H17M'),
               xs:dayTimeDuration('P3D'))"/>

    <xsl:text>&#xA;Here are some tests of the max() function:&#xA;</xsl:text>

    <xsl:text>&#xA;  max(</xsl:text>
    <xsl:value-of select="$seq1" separator=", "/>
    <xsl:text>) = </xsl:text>
    <xsl:value-of select="format-number(max($seq1), '#.###')"/>

    <xsl:text>&#xA;&#xA;  max(</xsl:text>
    <xsl:value-of select="$seq2" separator=", "/>
    <xsl:text>) = </xsl:text>
    <xsl:value-of select="format-number(max($seq2), '#.###')"/>

    <xsl:text>&#xA;&#xA;  max(</xsl:text>
    <xsl:value-of select="$seq3" separator=", "/>
    <xsl:text>) = </xsl:text>
    <xsl:value-of select="format-number(max($seq3), '#.###')"/>

    <xsl:text>&#xA;&#xA;  max(</xsl:text>
    <xsl:value-of select="$seq4" separator=", "/>
    <xsl:text>) = </xsl:text>
    <xsl:value-of select="max($seq4)"/>

    <xsl:text>&#xA;&#xA;    In text, the maximum of</xsl:text>
    <xsl:for-each select="$seq4">
```

```
      <xsl:text>&#xA;      </xsl:text>
      <xsl:value-of select="years-from-duration(.)"/>
      <xsl:text> years and </xsl:text>
      <xsl:value-of select="months-from-duration(.)"/>
      <xsl:text> months (</xsl:text>
      <xsl:value-of select="."/>
      <xsl:text>)</xsl:text>
    </xsl:for-each>

    <xsl:text>&#xA;    is </xsl:text>
    <xsl:value-of select="years-from-duration(max($seq4))"/>
    <xsl:text> years and </xsl:text>
    <xsl:value-of select="months-from-duration(max($seq4))"/>
    <xsl:text> months (</xsl:text>
    <xsl:value-of select="max($seq4)"/>
    <xsl:text>).</xsl:text>

    <xsl:text>&#xA;&#xA;  max(</xsl:text>
    <xsl:value-of select="$seq5" separator=", "/>
    <xsl:text>) = </xsl:text>
    <xsl:variable name="max5" select="max($seq5)"/>
    <xsl:value-of select="$max5"/>

    <xsl:text>&#xA;&#xA;    In text, the maximum of</xsl:text>
    <xsl:for-each select="$seq5">
      <xsl:text>&#xA;      </xsl:text>
      <xsl:value-of select="days-from-duration(.)"/>
      <xsl:text> days, </xsl:text>
      <xsl:value-of select="hours-from-duration(.)"/>
      <xsl:text> hours, </xsl:text>
      <xsl:value-of select="minutes-from-duration(.)"/>
      <xsl:text> minutes and </xsl:text>
      <xsl:value-of
        select="format-number(seconds-from-duration(.), '#.##')"/>
      <xsl:text> seconds (</xsl:text>
      <xsl:value-of select="."/>
      <xsl:text>)</xsl:text>
    </xsl:for-each>

    <xsl:text>&#xA;    is </xsl:text>
    <xsl:value-of select="days-from-duration($max5)"/>
    <xsl:text> days, </xsl:text>
    <xsl:value-of select="hours-from-duration($max5)"/>
    <xsl:text> hours, </xsl:text>
    <xsl:value-of select="minutes-from-duration($max5)"/>
    <xsl:text> minutes and </xsl:text>
    <xsl:value-of
      select="format-number(seconds-from-duration($max5), '#.##')"/>
    <xsl:text> seconds.</xsl:text>

  </xsl:template>

</xsl:stylesheet>
```

Результат применения таблицы стилей:

```
Here are some tests of the max() function:

  max(3, 5, 18) = 18

  max(3, 5, 48.273, 2900) = 2900
```

```
max(42, 2.71828, 98.6, 2.54) = 98.6

max(P3Y8M, P4Y2M, P6Y4M) = P6Y4M

  In text, the maximum of
    3 years and 8 months (P3Y8M)
    4 years and 2 months (P4Y2M)
    6 years and 4 months (P6Y4M)
  is 6 years and 4 months (P6Y4M).

max(P2DT4H23M12.2S, P3DT8H17M, P3D) = P3DT8H17M

  In text, the maximum of
    2 days, 4 hours, 23 minutes and 12.2 seconds (P2DT4H23M12.2S)
    3 days, 8 hours, 17 minutes and 0 seconds (P3DT8H17M)
    3 days, 0 hours, 0 minutes and 0 seconds (P3D)
  is 3 days, 8 hours, 17 minutes and 0 seconds.
```

Таблица стилей демонстрирует пять разных типов последовательностей. Первая последовательность содержит только целые числа, вторая является числовой последовательностью. Все значения в первых двух последовательностях преобразуются к числовому виду, поэтому результат соответствует ожиданиям. Третья последовательность содержит четыре числовых значения, каждое из которых относится к определенному числовому типу. Последние две последовательности состоят из двух типов временных промежутков. В таблице стилей используются также функции `years-from-duration()` и `format-number()` для форматирования данных.

Короткий пример с применением нестандартной схемы упорядочения:

```
<?xml version="1.0"?>
<!-- max2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="seq1"
      select="('strasse', 'stra&#xDF;e')"/>
    <xsl:variable name="seq2" select="reverse($seq1)"/>

    <xsl:text>&#xA;Here are some tests of the max() function:&#xA;</xsl:text>

    <xsl:text>&#xA;  max(</xsl:text>
    <xsl:value-of select="$seq1" separator=", "/>
    <xsl:text>) = </xsl:text>
    <xsl:value-of select="max($seq1)"/>

    <xsl:text>&#xA;  max(</xsl:text>
    <xsl:value-of select="$seq1" separator=", "/>
    <xsl:text> [German collation]) = </xsl:text>
    <xsl:value-of
      select="max($seq1,
               concat('http://saxon.sf.net/collation?',
                      'class=com.oreilly.xslt.GermanCollation;'))"/>

    <xsl:text>&#xA;  max(</xsl:text>
    <xsl:value-of select="$seq2" separator=", "/>
    <xsl:text> [German collation]) = </xsl:text>
```

```
    <xsl:value-of
      select="max($seq2,
              concat('http://saxon.sf.net/collation?',
                     'class=com.oreilly.xslt.GermanCollation;'))"/>

  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here are some tests of the max() function:

  max(strasse, straße) = straße
  max(strasse, straße [German collation]) = strasse
  max(straße, strasse [German collation]) = straße
```

При использовании стандартного упорядочения эти две строки не равны. В первом примере straße при сортировке размещается после strasse, поэтому функция max() возвращает именно straße. В двух последних примерах две строки при немецкой схеме упорядочения считаются равными. В Saxon 9.0.0.3J max() возвращает наибольшее значение, которое первым встречается в последовательности. Поведение в подобных случаях зависит от реализации; процессоры XSLT могут возвращать любое из равных наибольших значений.

[2.0] min()

Возвращает наименьшее значение из заданной последовательности.

Синтаксис

```
xs:anyAtomicType min(xs:anyAtomicType*)
xs:anyAtomicType min(xs:anyAtomicType*, $collation as xs:string)
```

Ввод

Последовательность значений. Необязательный аргумент, определяющий схему упорядочения, используется только в том случае, если последовательность содержит значения xs:string.

Вывод

Наименьшее значение из заданной последовательности. Функция min() работает со следующими типами значений: с числами (xs:integer, xs:double, xs:decimal и xs:float), с xs:string, xs:boolean, xs:date, xs:dateTime, xs:time, xs:yearMonthDuration и xs:dayTimeDuration (но *не* с xs:duration).

Для последовательности числовых значений процессор XSLT возвращает наименьшее число, преобразуя типы данных при необходимости. Для последовательности временных промежутков процессор XSLT возвращает промежуток наименьшей продолжительности. Для последовательности строк процессор XSLT возвращает строку, стоящую на первом месте в порядке сортировки.

Функция min() предполагает, что ей передается последовательность, содержащая осмысленные данные; в противном случае процессор XSLT выдает ошибку. Например, при попытке получения наименьшего значения в последовательности (xs:yearMonthDuration('P4Y2M'), 'strawberry', 3.14) произойдет ошибка – впрочем, как и следовало ожидать.

Несколько замечаний по поводу работы `min()`:

- Для поиска самого короткого временного промежутка в последовательности либо все значения должны относиться к типу `xs:dayTimeDuration`, либо все значения должны относиться к типу `xs:yearMonthDuration`. Смешивать эти две разновидности промежутков нельзя; при нарушении этого требования процессор XSLT выдает ошибку.
- Если в последовательности присутствуют значения типа `xs:untypedAtomic`, процессор XSLT пытается преобразовать их к типу `xs:double`. Если объект не удается преобразовать к `xs:double`, процессор XSLT выдает ошибку.
- При сравнении значений `xs:string` используется схема упорядочения по умолчанию, если только вы явно не зададите другую схему.
- Наконец, при передаче пустой последовательности функции `min()` она возвращает пустую последовательность. Хотя функции в этом случае не передаются никакие полезные данные, процессор XSLT не выдает ошибку.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.4 «Aggregate Functions».

Пример

Следующая таблица стилей находит наименьшее значение в последовательностях нескольких разных типов:

```
<?xml version="1.0"?>
<!-- min.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="seq1" select="(3, 5, 18)"/>
    <xsl:variable name="seq2" select="(3, 5, 48.273, 2.9e3)"/>

    <xsl:variable name="value1" as="xs:integer" select="42"/>
    <xsl:variable name="value2" as="xs:double" select="2718.28E-3"/>
    <xsl:variable name="value3" as="xs:float" select="98.6"/>
    <xsl:variable name="value4" as="xs:decimal" select="2.54"/>
    <xsl:variable name="seq3"
      select="($value1, $value2, $value3, $value4)"/>

    <xsl:variable name="seq4"
      select="(xs:yearMonthDuration('P3Y8M'),
               xs:yearMonthDuration('P4Y2M'),
               xs:yearMonthDuration('P6Y4M'))"/>
    <xsl:variable name="seq5"
      select="(xs:dayTimeDuration('P2DT4H23M12.2S'),
               xs:dayTimeDuration('P3DT8H17M'),
               xs:dayTimeDuration('P3D'))"/>
    <xsl:variable name="seq6"
      select="('red', 'white', 'blue')"/>

    <xsl:text>&#xA;Here are some tests of the min() function:&#xA;</xsl:text>
```

```
<xsl:text>&#xA;  min(</xsl:text>
<xsl:value-of select="$seq1" separator=", "/>
<xsl:text>) = </xsl:text>
<xsl:value-of select="format-number(min($seq1), '#.###')"/>

<xsl:text>&#xA;&#xA;  min(</xsl:text>
<xsl:value-of select="$seq2" separator=", "/>
<xsl:text>) = </xsl:text>
<xsl:value-of select="format-number(min($seq2), '#.###')"/>

<xsl:text>&#xA;&#xA;  min(</xsl:text>
<xsl:value-of select="$seq3" separator=", "/>
<xsl:text>) = </xsl:text>
<xsl:value-of select="format-number(min($seq3), '#.###')"/>

<xsl:text>&#xA;&#xA;  min(</xsl:text>
<xsl:value-of select="$seq4" separator=", "/>
<xsl:text>) = </xsl:text>
<xsl:value-of select="min($seq4)"/>

<xsl:text>&#xA;&#xA;    In text, the minimum of</xsl:text>
<xsl:for-each select="$seq4">
  <xsl:text>&#xA;      </xsl:text>
  <xsl:value-of select="years-from-duration(.)"/>
  <xsl:text> years and </xsl:text>
  <xsl:value-of select="months-from-duration(.)"/>
  <xsl:text> months (</xsl:text>
  <xsl:value-of select="."/>
  <xsl:text>)</xsl:text>
</xsl:for-each>

<xsl:text>&#xA;    is </xsl:text>
<xsl:value-of select="years-from-duration(min($seq4))"/>
<xsl:text> years and </xsl:text>
<xsl:value-of select="months-from-duration(min($seq4))"/>
<xsl:text> months (</xsl:text>
<xsl:value-of select="min($seq4)"/>
<xsl:text>).</xsl:text>

<xsl:text>&#xA;&#xA;  min(</xsl:text>
<xsl:value-of select="$seq5" separator=", "/>
<xsl:text>) = </xsl:text>
<xsl:variable name="min5" select="min($seq5)"/>
<xsl:value-of select="$min5"/>

<xsl:text>&#xA;&#xA;    In text, the minimum of</xsl:text>
<xsl:for-each select="$seq5">
  <xsl:text>&#xA;      </xsl:text>
  <xsl:value-of select="days-from-duration(.)"/>
  <xsl:text> days, </xsl:text>
  <xsl:value-of select="hours-from-duration(.)"/>
  <xsl:text> hours, </xsl:text>
  <xsl:value-of select="minutes-from-duration(.)"/>
  <xsl:text> minutes and </xsl:text>
  <xsl:value-of
    select="format-number(seconds-from-duration(.), '#.##')"/>
  <xsl:text> seconds (</xsl:text>
  <xsl:value-of select="."/>
  <xsl:text>)</xsl:text>
</xsl:for-each>
```

```
    <xsl:text>&#xA;    is </xsl:text>
    <xsl:value-of select="days-from-duration($min5)"/>
    <xsl:text> days, </xsl:text>
    <xsl:value-of select="hours-from-duration($min5)"/>
    <xsl:text> hours, </xsl:text>
    <xsl:value-of select="minutes-from-duration($min5)"/>
    <xsl:text> minutes and </xsl:text>
    <xsl:value-of
      select="format-number(seconds-from-duration($min5), '#.##')"/>
    <xsl:text> seconds.</xsl:text>

    <xsl:text>&#xA;&#xA;  min(</xsl:text>
    <xsl:value-of select="$seq6" separator=", "/>
    <xsl:text>) = </xsl:text>
    <xsl:value-of select="min($seq6)"/>

  </xsl:template>
</xsl:stylesheet>
```

Результат:

```
Here are some tests of the min() function:

  min(3, 5, 18) = 3

  min(3, 5, 48.273, 2900) = 3

  min(42, 2.71828, 98.6, 2.54) = 2.54

  min(P3Y8M, P4Y2M, P6Y4M) = P3Y8M

    In text, the minimum of
      3 years and 8 months (P3Y8M)
      4 years and 2 months (P4Y2M)
      6 years and 4 months (P6Y4M)
    is 3 years and 8 months (P3Y8M).

  min(P2DT4H23M12.2S, P3DT8H17M, P3D) = P2DT4H23M12.2S

    In text, the minimum of
      2 days, 4 hours, 23 minutes and 12.2 seconds (P2DT4H23M12.2S)
      3 days, 8 hours, 17 minutes and 0 seconds (P3DT8H17M)
      3 days, 0 hours, 0 minutes and 0 seconds (P3D)
    is 2 days, 4 hours, 23 minutes and 12.2 seconds.
```

Таблица стилей демонстрирует шесть разных типов последовательностей. Первая последовательность содержит только целые числа, вторая является числовой последовательностью. Все значения в первых двух последовательностях преобразуются к числовому виду, поэтому результат соответствует ожиданиям. Третья последовательность содержит четыре числовых значения, каждое из которых относится к определенному числовому типу. Следующие две последовательности состоят из двух типов временных промежутков, а последняя последовательность содержит строки. В таблице стилей используются также функции `years-from-duration()` и `format-number()` для форматирования данных.

Короткий пример с применением нестандартной схемы упорядочения:

```
<?xml version="1.0"?>
<!-- min2.xsl -->
<xsl:stylesheet version="2.0"
```

```
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xsl:output method="text"/>

    <xsl:template match="/">
      <xsl:variable name="seq1"
        select="('stra&#xDF;e', 'strasse')"/>
      <xsl:variable name="seq2" select="reverse($seq1)"/>

      <xsl:text>&#xA;Here are some tests of the min() function:&#xA;</xsl:text>

      <xsl:text>&#xA;  min(</xsl:text>
      <xsl:value-of select="$seq1" separator=", "/>
      <xsl:text>) = </xsl:text>
      <xsl:value-of select="min($seq1)"/>

      <xsl:text>&#xA;  min(</xsl:text>
      <xsl:value-of select="$seq1" separator=", "/>
      <xsl:text> [German collation]) = </xsl:text>
      <xsl:value-of
        select="min($seq1,
                concat('http://saxon.sf.net/collation?',
                       'class=com.oreilly.xslt.GermanCollation;'))"/>

      <xsl:text>&#xA;  min(</xsl:text>
      <xsl:value-of select="$seq2" separator=", "/>
      <xsl:text> [German collation]) = </xsl:text>
      <xsl:value-of
        select="min($seq2,
                concat('http://saxon.sf.net/collation?',
                       'class=com.oreilly.xslt.GermanCollation;'))"/>

    </xsl:template>

  </xsl:stylesheet>
```

Результат:

```
  Here are some tests of the min() function:

    min(strasse, straße) = strasse
    min(straße, strasse [German collation]) = straße
    min(strasse, straße [German collation]) = strasse
```

При использовании стандартного упорядочения эти две строки не равны. В первом примере `strasse` при сортировке размещается перед `straße`, поэтому функция `min()` возвращает именно `strasse`. В двух последних примерах две строки при немецкой схеме упорядочения считаются равными. В Saxon 9.0.0.3J `min()` возвращает наименьшее значение, которое первым встречается в последовательности. Поведение в подобных случаях зависит от реализации; процессоры XSLT могут возвращать любое из равных наименьших значений.

[2.0] minutes-from-dateTime()

Для заданного значения `xs:dateTime` возвращает минуты.

Синтаксис

```
xs:integer? minutes-from-dateTime(xs:dateTime?)
```

Ввод

Значение `xs:dateTime`.

Вывод

Значение `xs:integer`, представляющее компонент минут заданного значения `xs:dateTime`. Если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Таблица стилей для демонстрации возможностей функции `minutes-from-dateTime()`:

```
<?xml version="1.0"?>
<!-- minutes-from-datetime.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the minutes from an xs:dateTime:</xsl:text>
    <xsl:variable name="currentDateTime" as="xs:dateTime"
      select="current-dateTime()"/>
    <xsl:text>&#xA;&#xA;  The current date and time is: </xsl:text>
    <xsl:value-of select="$currentDateTime"/>

    <xsl:text>&#xA;&#xA;  The current minute is: </xsl:text>
    <xsl:value-of select="minutes-from-dateTime($currentDateTime)"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Extracting the minutes from an xs:dateTime:

  The current date and time is: 2006-11-16T04:34:19.921-05:00

  The current minute is: 34
```

См. также

Определения функций *[2.0]* `day-from-dateTime()`, *[2.0]* `format-dateTime()`, *[2.0]* `hours-from-dateTime()`, *[2.0]* `month-from-dateTime()`, *[2.0]* `seconds-from-dateTime()`, *[2.0]* `timezone-from-dateTime()` и *[2.0]* `year-from-dateTime()`.

[2.0] minutes-from-duration()

Возвращает количество минут во временном промежутке `xs:duration`.

Синтаксис

```
xs:integer? minutes-from-duration(xs:duration?)
```

Ввод

Значение `xs:duration`.

Вывод

Значение `xs:integer`, представляющее компонент минут в заданном промежутке `xs:duration`. Учтите, что для `xs:yearMonthDuration` эта функция всегда возвращает `0`, потому что `xs:yearMonthDuration` не содержит компонент минут. Кроме того, если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Следующая таблица стилей демонстрирует работу функции `minutes-from-duration()` для всех трех типов промежутков:

```
<?xml version="1.0"?>
<!-- minutes-from-duration.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the minutes component from durations:</xsl:text>

    <xsl:variable name="sampleDuration" as="xs:duration"
      select="xs:duration('P3Y8M2DT4H23M12.2S')"/>
    <xsl:variable name="sampleYearMonthDuration" as="xs:yearMonthDuration"
      select="xs:yearMonthDuration('P3Y8M')"/>
    <xsl:variable name="sampleDayTimeDuration" as="xs:dayTimeDuration"
      select="xs:dayTimeDuration('P2DT4H23M12.2S')"/>

    <xsl:text>&#xA;&#xA;  A sample xs:duration: </xsl:text>
    <xsl:value-of select="$sampleDuration"/>
    <xsl:text>&#xA;    The minutes component of this duration is </xsl:text>
    <xsl:value-of select="minutes-from-duration($sampleDuration)"/>
    <xsl:text>.</xsl:text>

    <xsl:text>&#xA;&#xA;  A sample xs:yearMonthDuration: </xsl:text>
    <xsl:value-of select="$sampleYearMonthDuration"/>
    <xsl:text>&#xA;    The minutes component of this duration is </xsl:text>
    <xsl:value-of select="minutes-from-duration($sampleYearMonthDuration)"/>
    <xsl:text>.</xsl:text>

    <xsl:text>&#xA;&#xA;  A sample xs:dayTimeDuration: </xsl:text>
    <xsl:value-of select="$sampleDayTimeDuration"/>
    <xsl:text>&#xA;    The minutes component of this duration is </xsl:text>
    <xsl:value-of select="minutes-from-duration($sampleDayTimeDuration)"/>
    <xsl:text>.</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Extracting the minutes component from durations:

  A sample xs:duration: P3Y8M2DT4H23M12.2S
    The minutes component of this duration is 23.

  A sample xs:yearMonthDuration: P3Y8M
    The minutes component of this duration is 0.

  A sample xs:dayTimeDuration: P2DT4H23M12.2S
    The minutes component of this duration is 23.
```

Как и следовало ожидать, вызов `minutes-from-duration()` для значения `xs:yearMonthDuration` возвращает `0`.

См. также

Определения функций *[2.0]* `days-from-duration()`, *[2.0]* `hours-from-duration()`, *[2.0]* `months-from-duration()`, *[2.0]* `seconds-from-duration()` и *[2.0]* `years-from-duration()`.

[2.0] minutes-from-time()

Для заданного значения `xs:time` возвращает минуты.

Синтаксис

```
xs:integer? minutes-from-time(xs:time?)
```

Ввод

Значение `xs:time`.

Вывод

Значение `xs:integer`, представляющее компонент минут заданного значения `xs:time`. Если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Таблица стилей для демонстрации работы функции `minutes-from-time()`:

```
<?xml version="1.0"?>
<!-- minutes-from-time.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the minutes component from an xs:time:</xsl:text>
    <xsl:variable name="currentTime" as="xs:time" select="current-time()"/>
    <xsl:text>&#xA;&#xA;  The current time is: </xsl:text>
```

```
    <xsl:value-of select="$currentTime"/>

    <xsl:text>&#xA;&#xA;  The current minute is: </xsl:text>
    <xsl:value-of select="minutes-from-time($currentTime)"/>
    <xsl:text>, the </xsl:text>
    <xsl:value-of select="format-time($currentTime, '[mwo]')"/>
    <xsl:text> minute of the hour.</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Extracting the minutes component from an xs:time:

  The current time is: 04:39:49.875-05:00

  The current minute is: 39, the thirty-ninth minute of the hour.
```

Первый результат сгенерирован функцией `minutes-from-time()`, остальные – функцией `format-time()` с форматной строкой, выбирающей компонент минут из значения `xs:time`.

См. также

Определения функций *[2.0]* `format-time()`, *[2.0]* `hours-from-time()`, *[2.0]* `seconds-from-time()` и *[2.0]* `timezone-from-time()`.

[2.0] month-from-date()

Возвращает компонент месяца для полученного значения `xs:date`.

Синтаксис

```
xs:integer? month-from-date(xs:date?)
```

Ввод

Значение `xs:date`.

Вывод

Значение `xs:integer`, представляющее компонент месяца для заданного значения `xs:date`. Если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Для демонстрации работы функции `month-from-date()` используется следующая таблица стилей:

```
<?xml version="1.0"?>
<!-- month-from-date.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xsl:output method="text"/>

    <xsl:template match="/">
      <xsl:text>&#xA;Extracting the month from an xs:date:</xsl:text>
      <xsl:variable name="currentDate" as="xs:date" select="current-date()"/>
      <xsl:text>&#xA;&#xA;The current date is: </xsl:text>
      <xsl:value-of select="$currentDate"/>

      <xsl:text>&#xA;&#xA;  The current month: </xsl:text>
      <xsl:value-of select="month-from-date($currentDate)"/>
      <xsl:text>&#xA;    In English: </xsl:text>
      <xsl:value-of select="format-date($currentDate, '[MNn]')"/>
      <xsl:text>&#xA;    In German: </xsl:text>
      <xsl:value-of select="format-date($currentDate, '[MNn]', 'de', (), ())"/>
      <xsl:text>&#xA;    It's the </xsl:text>
      <xsl:value-of select="format-date($currentDate, '[M1o]')"/>
      <xsl:text> month of the year.</xsl:text>
    </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Extracting the month from an xs:dateTime:

The current date and time is: 2006-07-05T12:38:00.877-05:00

  The current month: 7
    In English: July
    In German: Juli
    It's the 7th month of the year.
```

Обратите внимание: первый результат генерируется функцией `month-from-date()`**, а другие – функцией** `format-date()` **с форматной строкой, выбирающей только компонент месяца из значения** `xs:date`**.**

См. также

Определения функций ***[2.0]*** `day-from-date()`**,** ***[2.0]*** `format-date()`**,** ***[2.0]*** `timezone-from-date()` **и** ***[2.0]*** `year-from-date()`**.**

[2.0] month-from-dateTime()

Возвращает компонент месяца для полученного значения `xs:dateTime`**.**

Синтаксис

```
xs:integer? month-from-dateTime(xs:date?)
```

Ввод

Значение `xs:dateTime`**.**

Вывод

Значение `xs:integer`**, представляющее компонент месяца для заданного значения** `xs:dateTime`**. Если аргумент является пустой последовательностью, функция возвращает пустую последовательность.**

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Для демонстрации работы функции `month-from-dateTime()` используется следующая таблица стилей:

```
<?xml version="1.0"?>
<!-- month-from-datetime.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the month from an xs:dateTime:</xsl:text>
    <xsl:variable name="currentDateTime" as="xs:dateTime"
      select="current-dateTime()"/>
    <xsl:text>&#xA;&#xA;The current date and time is: </xsl:text>
    <xsl:value-of select="$currentDateTime"/>

    <xsl:text>&#xA;&#xA;  The current month: </xsl:text>
    <xsl:value-of select="month-from-dateTime($currentDateTime)"/>
    <xsl:text>&#xA;    In English: </xsl:text>
    <xsl:value-of select="format-dateTime($currentDateTime, '[MNn]')"/>
    <xsl:text>&#xA;    In German: </xsl:text>
    <xsl:value-of select="format-dateTime($currentDateTime, '[MNn]', 'de', (), ())"/>
    <xsl:text>&#xA;    It's the </xsl:text>
    <xsl:value-of select="format-dateTime($currentDateTime, '[M1o]')"/>
    <xsl:text> month of the year.</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Extracting the month from an xs:dateTime:

The current date and time is: 2006-11-16T04:42:47.481-05:00

  The current month: 11
    In English: November
    In German: November
    It's the 11th month of the year.
```

Обратите внимание: одни результаты генерируются функцией `month-from-dateTime()`, а другие — функцией `format-dateTime()` с форматной строкой, выбирающей только компонент месяца из значения `xs:dateTime`.

См. также

Определения функций *[2.0]* `day-from-dateTime()`, *[2.0]* `format-dateTime()`, *[2.0]* `hours-from-dateTime()`, *[2.0]* `minutes-from-dateTime()`, *[2.0]* `seconds-from-dateTime()`, *[2.0]* `timezone-from-dateTime()` и *[2.0]* `year-from-dateTime()`.

[2.0] months-from-duration()

Возвращает количество месяцев в заданном временном промежутке `xs:duration`.

Синтаксис

```
xs:integer? months-from-duration(xs:duration?)
```

Ввод

Значение `xs:duration`.

Вывод

Значение `xs:integer`, представляющее количество месяцев в заданном промежутке `xs:duration`. Учтите, что для `xs:dayTimeDuration` эта функция всегда возвращает `0`, потому что в значении `xs:dayTimeDuration` отсутствует компонент месяцев. Кроме того, если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Следующая таблица стилей демонстрирует использование функции `months-from-duration()` для всех трех типов временных промежутков:

```
<?xml version="1.0"?>
<!-- months-from-duration.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the months component from durations:</xsl:text>

    <xsl:variable name="sampleDuration" as="xs:duration"
      select="xs:duration('P3Y8M2DT4H23M12.2S')"/>
    <xsl:variable name="sampleYearMonthDuration" as="xs:yearMonthDuration"
      select="xs:yearMonthDuration('P3Y8M')"/>
    <xsl:variable name="sampleDayTimeDuration" as="xs:dayTimeDuration"
      select="xs:dayTimeDuration('P2DT4H23M12.2S')"/>

    <xsl:text>&#xA;&#xA;  A sample xs:duration: </xsl:text>
    <xsl:value-of select="$sampleDuration"/>
    <xsl:text>&#xA;    The months component of this duration is </xsl:text>
    <xsl:value-of select="months-from-duration($sampleDuration)"/>
    <xsl:text>.</xsl:text>

    <xsl:text>&#xA;&#xA;  A sample xs:yearMonthDuration: </xsl:text>
    <xsl:value-of select="$sampleYearMonthDuration"/>
    <xsl:text>&#xA;    The months component of this duration is </xsl:text>
    <xsl:value-of select="months-from-duration($sampleYearMonthDuration)"/>
    <xsl:text>.</xsl:text>
```

```
      <xsl:text>&#xA;&#xA;  A sample xs:dayTimeDuration: </xsl:text>
      <xsl:value-of select="$sampleDayTimeDuration"/>
      <xsl:text>&#xA;    The months component of this duration is </xsl:text>
      <xsl:value-of select="months-from-duration($sampleDayTimeDuration)"/>
      <xsl:text>.</xsl:text>
    </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Extracting the months component from durations:

  A sample xs:duration: P3Y8M2DT4H23M12.2S
    The months component of this duration is 8.

  A sample xs:yearMonthDuration: P3Y8M
    The months component of this duration is 8.

  A sample xs:dayTimeDuration: P2DT4H23M12.2S
    The months component of this duration is 0.
```

Как видно из результата, при извлечении месяцев из `xs:dayTimeDuration` возвращается `0`.

См. также

Определения функций *[2.0]* `days-from-duration()`, *[2.0]* `hours-from-duration()`, *[2.0]* `minutes-from-duration()`, *[2.0]* `seconds-from-duration()` и *[2.0]* `years-from-duration()`.

name()

Возвращает полное имя узла. Полное (уточненное) имя включает префикс пространства имен. Для получения информации об URI-адресе (не префиксе!) пространства имен XPath предоставляет функцию `namespace-uri()`.

Синтаксис

```
[1.0] string name(node-set?)
[2.0] xs:string name(node()?)
```

Ввод

[1.0] Необязательный набор узлов. Если набор узлов задан, процессор XSLT 1.0 вызывает функцию `name()` для первого узла в наборе. Если набор узлов не задан, функция `name()` применяется к контекстному узлу.

[2.0] Необязательная последовательность узлов. *В XSLT 2.0 вызов* `name()` *для последовательности, содержащей более одного узла, является ошибкой*. Если последовательность узлов не задана, функция `name()` применяется к контекстному узлу.

Вывод

Полное имя узла. Если аргумент функции `name()` пуст (скажем, при использовании атрибута `select="item"`, когда в текущем контексте нет ни одного элемента `<item>`), `name()` возвращает пустую строку.

Где определяется

[1.0] XPath, раздел 4.1 «Node Set Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 14 «Functions and Operators on Nodes».

Пример

Документ XML, который будет использоваться для демонстрации работы функции `name()`:

```
<?xml version="1.0"?>
<!-- sonnet.xml -->
<sonnet type='Shakespearean'>
  <auth:author xmlns:auth="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </auth:author>
  <!-- Существует ли официальное название сонета?
       Иногда сонеты называются по первой строке. -->
  <title>Sonnet 130</title>
  <lines>
    <line>My mistress' eyes are nothing like the sun,</line>
    <line>Coral is far more red than her lips red.</line>
    <line>If snow be white, why then her breasts are dun,</line>
    <line>If hairs be wires, black wires grow on her head.</line>
    <line>I have seen roses damasked, red and white,</line>
    <line>But no such roses see I in her cheeks.</line>
    <line>And in some perfumes is there more delight</line>
    <line>Than in the breath that from my mistress reeks.</line>
    <line>I love to hear her speak, yet well I know</line>
    <line>That music hath a far more pleasing sound.</line>
    <line>I grant I never saw a goddess go,</line>
    <line>My mistress when she walks, treads on the ground.</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
  </lines>
</sonnet>
```

Следующая таблица стилей выводит результат вызова функции `name()` для каждого узла в документе XML:

```
<?xml version="1.0"?>
<!-- name.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the name() function:&#xA;</xsl:text>

    <xsl:for-each select="//*">
      <xsl:text>&#xA;  Element </xsl:text>
      <xsl:number level="any" count="*"/>
```

```
      <xsl:text>: </xsl:text>
      <xsl:value-of select="name()"/>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Преобразование документа XML по этой таблице стилей дает следующий результат:

```
A test of the name() function:

  Element 1: sonnet
  Element 2: auth:author
  Element 3: last-name
  Element 4: first-name
  Element 5: nationality
  Element 6: year-of-birth
  Element 7: year-of-death
  Element 8: title
  Element 9: lines
  Element 10: line
  Element 11: line
  Element 12: line
  Element 13: line
  Element 14: line
  Element 15: line
  Element 16: line
  Element 17: line
  Element 18: line
  Element 19: line
  Element 20: line
  Element 21: line
  Element 22: line
  Element 23: line
```

Далее приводится чуть более сложная версия, в которой элемент XSLT 2.0 <xsl:sequence> выбирает последовательность узлов документа. Последовательность сохраняется в переменной, после чего таблица стилей перебирает значения из переменной и выводит имена элементов:

```
<?xml version="1.0"?>
<!-- name2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.oreilly.com/xslt"
  xmlns:auth="http://www.authors.com/">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the name() function:&#xA;</xsl:text>

    <xsl:variable name="authorDetails" as="element()*"
      select="/sonnet/auth:author/*"/>
    <xsl:for-each select="$authorDetails">
      <xsl:text>&#xA;  Element </xsl:text>
      <xsl:value-of select="position()"/>
      <xsl:text>: </xsl:text>
      <xsl:value-of select="name()"/>
```

```
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

(Обратите внимание на сохранение последовательности в переменной с типом `element()*`, а также на необходимость определения пространства имен `auth` в таблице стилей.) Результат содержит имена всех элементов, находящихся ниже элемента `<auth:author>` в исходном документе:

```
Tests of the name() function:

  Element 1: last-name
  Element 2: first-name
  Element 3: nationality
  Element 4: year-of-birth
  Element 5: year-of-death
```

Как упоминалось ранее, в XSLT 2.0 передача более одного узла функции `name()` является ошибкой. (В XSLT 1.0 процессор просто использует первый узел в наборе узлов и игнорирует все остальное.) Функции `name()` можно передать переменную с типом `element()*`, но эта последовательность не может содержать более одного объекта. Например, следующий фрагмент допустим в XSLT 2.0:

```
<xsl:variable name="salesFigures" as="element()*"
  select="/report/brand[1]/*[2]"/>
<xsl:value-of select="name($salesFigures)"/>
```

Он допустим, потому что выражение XPath в элементе `<xsl:variable>` выбирает только один узел. Однако со следующим фрагментом процессор XSLT 2.0 выдает ошибку:

```
<xsl:variable name="salesFigures" as="element()*"
  select="/report/brand[1]/*"/>
<xsl:value-of select="name($salesFigures)"/>
```

Ошибка возникает из-за того, что выражение XPath выбирает более одного узла в исходном документе. Если изменить исходный документ XML так, чтобы под первым элементом `<brand>` остался только один узел, таблица стилей будет работать.

namespace-uri()

Возвращает URI-адрес пространства имен для узла, переданного в аргументе.

Синтаксис

```
[1.0] string namespace-uri(node-set?)
[2.0] xs:anyURI namespace-uri()
[2.0] xs:anyURI namespace-uri(node()?)
```

Ввод

[1.0] Набор узлов. Если набор узлов не задан, функция `namespace-uri()` создает набор узлов, состоящий только из контекстного узла.

[2.0] Узел. Если узел не указан, функция `namespace-uri()` применяется к контекстному узлу. Если входные данные представляют собой последовательность, содержащую более одного узла, процессор XSLT выдает ошибку.

Вывод

[1.0] URI-адрес пространства имен для первого узла в наборе узлов, переданном в аргументе. Если набор узлов пуст, первый узел не имеет URI-адреса пространства имен или URI-адрес пространства имен первого узла пуст, функция возвращает пустую строку.

[2.0] URI-адрес пространства имен узла, переданного в аргументе. Если аргумент является пустой последовательностью, аргумент не имеет URI-адреса пространства имен или URI-адрес пространства имен узла пуст, функция возвращает пустую строку.

Учтите, что для всех остальных узлов, кроме узлов элементов и атрибутов, функция `namespace-uri()` возвращает пустую строку.

Где определяется

[1.0] XPath, раздел 4.1 «Node Set Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 14 «Functions and Operators on Nodes».

Пример

Для демонстрации работы функции `namespace-uri()` будет использоваться слегка измененный документ с шекспировским сонетом:

```
<?xml version="1.0"?>
<!-- sonnet-namespace.xml -->
<sonnet type='Shakespearean'
  xmlns="http://www.oreilly.com/xslt">
  <author xmlns="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </author>
  <!-- Существует ли официальное название сонета?
       Иногда сонеты называются по первой строке. -->
  <title>Sonnet 130</title>
  <lines>
    <line>My mistress' eyes are nothing like the sun,</line>
    <line>Coral is far more red than her lips red.</line>
    <line>If snow be white, why then her breasts are dun,</line>
    <line>If hairs be wires, black wires grow on her head.</line>
    <line>I have seen roses damasked, red and white,</line>
    <line>But no such roses see I in her cheeks.</line>
    <line>And in some perfumes is there more delight</line>
    <line>Than in the breath that from my mistress reeks.</line>
    <line>I love to hear her speak, yet well I know</line>
    <line>That music hath a far more pleasing sound.</line>
    <line>I grant I never saw a goddess go,</line>
    <line>My mistress when she walks, treads on the ground.</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
  </lines>
</sonnet>
```

Документ был изменен таким образом, чтобы он имел пространство имен по умолчанию (`http://www.oreilly.com/`**). Наша таблица стилей перебирает содержимое документа XML и вызывает** `namespace-uri()` **для всех узлов элементов и атрибутов в документе:**

```
<?xml version="1.0"?>
<!-- namespace-uri.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the namespace-uri() function:&#xA;&#xA;</xsl:text>

    <xsl:for-each select="//*">
      <xsl:text>  Namespace URI for &lt;</xsl:text>
      <xsl:value-of select="node-name(.)"/>
      <xsl:text>&gt;: </xsl:text>
      <xsl:value-of select="namespace-uri()"/>
      <xsl:text>&#xA;</xsl:text>
      <xsl:for-each select="@*">
        <xsl:text>    Namespace URI for attribute "</xsl:text>
        <xsl:value-of select="node-name(.)"/>
        <xsl:text>": </xsl:text>
        <xsl:value-of select="namespace-uri()"/>
        <xsl:text>&#xA;</xsl:text>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Результат применения таблицы стилей:

```
A test of the namespace-uri() function:

  Namespace URI for <sonnet>: http://www.oreilly.com/xslt
    Namespace URI for attribute "type":
  Namespace URI for <author>: http://www.authors.com/
  Namespace URI for <last-name>: http://www.authors.com/
  Namespace URI for <first-name>: http://www.authors.com/
  Namespace URI for <nationality>: http://www.authors.com/
  Namespace URI for <year-of-birth>: http://www.authors.com/
  Namespace URI for <year-of-death>: http://www.authors.com/
  Namespace URI for <title>: http://www.oreilly.com/xslt
  Namespace URI for <lines>: http://www.oreilly.com/xslt
  Namespace URI for <line>: http://www.oreilly.com/xslt
  Namespace URI for <line>: http://www.oreilly.com/xslt
  Namespace URI for <line>: http://www.oreilly.com/xslt
  Namespace URI for <line>: http://www.oreilly.com/xslt
  Namespace URI for <line>: http://www.oreilly.com/xslt
  Namespace URI for <line>: http://www.oreilly.com/xslt
  Namespace URI for <line>: http://www.oreilly.com/xslt
  Namespace URI for <line>: http://www.oreilly.com/xslt
  Namespace URI for <line>: http://www.oreilly.com/xslt
  Namespace URI for <line>: http://www.oreilly.com/xslt
  Namespace URI for <line>: http://www.oreilly.com/xslt
  Namespace URI for <line>: http://www.oreilly.com/xslt
```

```
Namespace URI for <line>: http://www.oreilly.com/xslt
Namespace URI for <line>: http://www.oreilly.com/xslt
```

В измененном сонете элемент <author> объявляет новое пространство имен по умолчанию, которое возвращается для всех элементов внутри него. За пределами элемента <author> снова начинает действовать исходное пространство имен по умолчанию.

[2.0] namespace-uri-for-prefix()

Для заданных префикса пространства имен и элемента возвращает URI-адрес, связанный с этим префиксом.

Синтаксис

```
xs:anyURI? namespace-uri-for-prefix($prefix as xs:string?,
                                    $element as element())
```

Ввод

Префикс пространства имен и элемент. Если префикс является пустой строкой, возвращается URI-адрес пространства имен по умолчанию (при условии, что это пространство находится в области видимости).

Вывод

URI-адрес, связанный с пространством имен заданного элемента.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 11.2 «Operators and Functions Related to QNames».

Пример

В качестве исходного документа XML снова используется сонет Шекспира:

```
<?xml version="1.0"?>
<!-- sonnet.xml -->
<sonnet type='Shakespearean'>
  <auth:author xmlns:auth="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </auth:author>
  <!-- Существует ли официальное название сонета?
       Иногда сонеты называются по первой строке. -->
  <title>Sonnet 130</title>
  <lines>
    <line>My mistress' eyes are nothing like the sun,</line>
    <line>Coral is far more red than her lips red.</line>
    <line>If snow be white, why then her breasts are dun,</line>
    <line>If hairs be wires, black wires grow on her head.</line>
    <line>I have seen roses damasked, red and white,</line>
    <line>But no such roses see I in her cheeks.</line>
    <line>And in some perfumes is there more delight</line>
    <line>Than in the breath that from my mistress reeks.</line>
```

```
    <line>I love to hear her speak, yet well I know</line>
    <line>That music hath a far more pleasing sound.</line>
    <line>I grant I never saw a goddess go,</line>
    <line>My mistress when she walks, treads on the ground.</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
  </lines>
</sonnet>
```

Таблица стилей:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- namespace-uri-for-prefix.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the namespace-uri-for-prefix() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  namespace-uri-for-prefix</xsl:text>
    <xsl:text>('auth', //last-name) = "</xsl:text>
    <xsl:value-of
      select="namespace-uri-for-prefix('auth', //last-name)"/>
    <xsl:text>"</xsl:text>

    <xsl:text>&#xA;  namespace-uri-for-prefix</xsl:text>
    <xsl:text>('auth', //*:author) = "</xsl:text>
    <xsl:value-of
      select="namespace-uri-for-prefix('auth', //*:author)"/>
    <xsl:text>"</xsl:text>

    <xsl:text>&#xA;  namespace-uri-for-prefix</xsl:text>
    <xsl:text>('auth', /sonnet) = "</xsl:text>
    <xsl:value-of
      select="namespace-uri-for-prefix('auth', /sonnet)"/>
    <xsl:text>"</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the namespace-uri-for-prefix() function:

  namespace-uri-for-prefix('auth', //last-name) = "http://www.authors.com/"
  namespace-uri-for-prefix('auth', //*:author) = "http://www.authors.com/"
  namespace-uri-for-prefix('auth', /sonnet) = ""
```

В этой таблице стилей мы ищем URI-адрес для префикса `auth`**. В первом тесте функции** `namespace-uri-for-prefix()` **префикс** `auth` **находится в области видимости элемента** `<auth:author>` **и всего, что находится внутри, поэтому XSLT возвращает URL-адрес, связанный с ним. Во втором тесте используется выражение XPath с метасимволом. (Для вызова функции** `namespace-uri-for-prefix('auth', //auth:author)` **пространство имен** `auth` **должно быть определено в таблице стилей.) Наконец, последний пример возвращает пустую строку, потому что префикс** `auth` **не входит в область видимости элемента** `<sonnet>`**.**

При написании этой таблицы стилей была известна информация о структуре документа XML. Вероятно, в более замысловатом примере функция `in-scope-prefixes()` получала бы последовательность префиксов, а затем использовала бы функцию `namespace-uri-for-prefix()` для получения URI-адресов, связанных с каждым префиксом.

Существуют и другие сложности: передача во втором аргументе пустой последовательности или последовательности, содержащей более одного объекта, приводит к фатальной ошибке. В данном примере проблем нет – при написании таблицы стилей было известно, что документ содержит элементы `<last-name>`, `<auth:author>` и `<sonnet>`, причем каждый из них встречается только один раз.

Как обычно, читатель может самостоятельно создать более надежную реализацию для развлечения.

[2.0] namespace-uri-from-QName()

Для заданного значения `xs:QName` возвращает URI-адрес пространства имен, связанного с этим полным именем.

Синтаксис

```
xs:anyURI? namespace-uri-from-QName(xs:QName?)
```

Ввод

Значение `xs:QName`.

Вывод

URI-адрес пространства имен, связанного с полным именем. Если аргумент является пустой последовательностью, функция возвращает пустую последовательность. Если полное имя не принадлежит пространству имен, возвращается строка нулевой длины.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 11.2 «Operators and Functions Related to QNames».

Пример

В качестве исходного документа XML снова используется сонет Шекспира:

```
<?xml version="1.0"?>
<!-- sonnet.xml -->
<sonnet type='Shakespearean'>
  <auth:author xmlns:auth="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </auth:author>
  <!-- Существует ли официальное название сонета?
       Иногда сонеты называются по первой строке. -->
  <title>Sonnet 130</title>
  <lines>
```

```
    <line>My mistress' eyes are nothing like the sun,</line>
    <line>Coral is far more red than her lips red.</line>
    <line>If snow be white, why then her breasts are dun,</line>
    <line>If hairs be wires, black wires grow on her head.</line>
    <line>I have seen roses damasked, red and white,</line>
    <line>But no such roses see I in her cheeks.</line>
    <line>And in some perfumes is there more delight</line>
    <line>Than in the breath that from my mistress reeks.</line>
    <line>I love to hear her speak, yet well I know</line>
    <line>That music hath a far more pleasing sound.</line>
    <line>I grant I never saw a goddess go,</line>
    <line>My mistress when she walks, treads on the ground.</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
  </lines>
</sonnet>
```

Таблица стилей:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- namespace-uri-from-QName.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the namespace-uri-from-QName() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:variable name="testQName1" as="xs:QName"
      select="resolve-QName('auth:last-name',
              /sonnet/*:author/last-name)"/>
    <xsl:text>&#xA;  testQName1 = resolve-QName('auth:last-name', </xsl:text>
    <xsl:text>/sonnet/*:author/last-name)</xsl:text>
    <xsl:text>&#xA;  namespace-uri-from-QName($testQName1) = "</xsl:text>
    <xsl:value-of select="namespace-uri-from-QName($testQName1)"/>
    <xsl:text>"</xsl:text>

    <xsl:variable name="testQName2" as="xs:QName"
      select="resolve-QName('something-else', /sonnet)"/>
    <xsl:text>&#xA;&#xA;  testQName2 = resolve-QName('</xsl:text>
    <xsl:text>something-else', /sonnet)</xsl:text>
    <xsl:text>&#xA;  namespace-uri-from-QName($testQName2) = "</xsl:text>
    <xsl:value-of select="namespace-uri-from-QName($testQName2)"/>
    <xsl:text>"</xsl:text>

    <xsl:variable name="testQName3" as="xs:QName"
      select="QName('http://www.authors.com/', 'pfx:writer')"/>
    <xsl:text>&#xA;&#xA;  testQName3 = QName('http://www.</xsl:text>
    <xsl:text>authors.com/', 'pfx:writer')&#xA;</xsl:text>
    <xsl:text>  namespace-uri-from-QName($testQName3) = "</xsl:text>
    <xsl:value-of select="namespace-uri-from-QName($testQName3)"/>
    <xsl:text>"</xsl:text>

  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the namespace-uri-from-QName() function:

  testQName1 = resolve-QName('auth:last-name', /sonnet/*:author/last-name)
  namespace-uri-from-QName($testQName1) = "http://www.authors.com/"

  testQName2 = resolve-QName('something-else', /sonnet)
  namespace-uri-from-QName($testQName2) = ""

  testQName3 = QName('http://www.authors.com/', 'pfx:writer')
  namespace-uri-from-QName($testQName3) = "http://www.authors.com/"
```

Аргумент функции `namespace-uri-from-QName()` *должен* относиться к типу `xs:QName`. В приведенной таблице стилей полные имена конструируются функцией `resolve-QName()` и функцией-конструктором `QName()`.

В первом тесте для создания `QName` используется полное имя элемента `auth:lastname`. Во втором аргументе `resolve-QName()` передается элемент `<last-name>` нашего сонета. Функция `resolve-QName()` возвращает полное имя с локальным именем `last-name`, префиксом пространства имен `auth` и URI-адресом пространства имен `http://www.authors.com/`. Если ни один узел не соответствует выражению XPath `/sonnet/*:author/last-name`, вызов `resolve-QName()` завершается неудачей. Сбой происходит и в том случае, если для элемента `<last-name>` не определен префикс `auth`.

Во втором тесте `namespace-uri-from-QName()` при вызове `resolve-QName()` используется имя элемента без префикса. Документ XML не имеет пространства имен по умолчанию, поэтому в полном имени префикс и URI-адрес пространства имен представлены пустыми строками.

В последнем тесте функция `QName()` создает полное имя «с нуля». В двух аргументах функции передаются URI-адрес пространства имен и имя элемента – в данном случае с префиксом. Полученное полное имя образуется из локального имени `writer`, префикса пространства имен `pfx` и URI-адреса пространства имен `http://www.authors.com/`.

Мы рассмотрим еще пару примеров, но на этот раз воспользуемся версией сонета, *имеющей* пространство имен по умолчанию:

```
<?xml version="1.0"?>
<!-- sonnet-default-namespace.xml -->
<sonnet type='Shakespearean'
  xmlns="http://www.oreilly.com/xslt">
  <auth:author xmlns:auth="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
...
</sonnet>
```

В новой версии сонета используется пространство имен по умолчанию `http://www.oreilly.com/xslt`. Теперь необходимо внести изменения в таблицу стилей:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- namespace-uri-from-QName2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>
```

```
  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the namespace-uri-from-QName() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:variable name="testQName1" as="xs:QName"
      select="resolve-QName('auth:last-name',
              /*:sonnet/*:author/*:last-name)"/>
    <xsl:text>&#xA;  testQName1 = resolve-QName('auth:last-name', </xsl:text>
    <xsl:text>/*:sonnet/*:author/*:last-name)</xsl:text>
    <xsl:text>&#xA;  namespace-uri-from-QName($testQName1) = "</xsl:text>
    <xsl:value-of select="namespace-uri-from-QName($testQName1)"/>
    <xsl:text>"</xsl:text>

    <xsl:variable name="testQName2" as="xs:QName"
      select="resolve-QName('something-else', /*:sonnet)"/>
    <xsl:text>&#xA;&#xA;  testQName2 = resolve-QName('</xsl:text>
    <xsl:text>something-else', /*:sonnet)</xsl:text>
    <xsl:text>&#xA;  namespace-uri-from-QName($testQName2) = "</xsl:text>
    <xsl:value-of select="namespace-uri-from-QName($testQName2)"/>
    <xsl:text>"</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Результат тоже слегка изменяется:

```
Tests of the namespace-uri-from-QName() function:

  testQName1 = resolve-QName('auth:last-name', /*:sonnet/*:author/*:last-name)
  namespace-uri-from-QName($testQName1) = "http://www.authors.com/"

  testQName2 = resolve-QName('something-else', /*:sonnet)
  namespace-uri-from-QName($testQName2) = "http://www.oreilly.com/xslt"
```

Различия в результатах объясняются тем, что мы получаем URI-адрес пространства имен для полного имени, созданного без префикса. У видоизмененного сонета имеется пространство имен по умолчанию, которое мы и получаем.

Что касается изменений в таблице стилей, нам приходится использовать звездочки (*) для ссылок на объекты из пространства имен `http://www.oreilly.com/xslt`**. Другими словами,** `/sonnet/*:auth/last-name` **приходится заменять на** `/*:sonnet/*:auth/*:last-name`**.**

Существует и другое решение: назначить префикс пространству имен по умолчанию в таблице стилей:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- namespace-uri-from-QName3.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:ora="http://www.oreilly.com/xslt">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the namespace-uri-from-QName() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:variable name="testQName1" as="xs:QName"
      select="resolve-QName('auth:last-name',
              /ora:sonnet/*:author/ora:last-name)"/>
  ...
```

```
  </xsl:template>

</xsl:stylesheet>
```

Полученный результат близок к результату второй таблицы стилей:

```
Tests of the namespace-uri-from-QName() function:

  testQName1 = resolve-QName('auth:last-name',
                             /ora:sonnet/*:author/ora:last-name)
  namespace-uri-from-QName($testQName1) = "http://www.authors.com/"

  testQName2 = resolve-QName('something-else', /ora:sonnet)
  namespace-uri-from-QName($testQName2) = "http://www.oreilly.com/xslt"
```

Префикс `ora` связывается с пространством имен по умолчанию в исходном документе XML. Везде, где в таблице стилей используется префикс `ora`, он обозначает пространство имен `http://www.oreilly.com/xslt`.

[2.0 – Schema] nilled()

Возвращает `true`, если заданный элемент обладает неопределенным значением, или `false` в противном случае. Если аргумент не является элементом, `nilled()` возвращает пустую последовательность.

Синтаксис

```
xs:boolean? nilled(node()?)
```

Ввод

Узел.

Вывод

Если заданный узел обладает неопределенным значением (то есть имеет атрибут `xsi:nil="true"`), функция возвращает `true`; в противном случае она возвращает `false`. При вызове `nilled()` для пустой последовательности возвращается пустая последовательность.

Функция `nilled()` работает только в схемосовместимых процессорах и только в том случае, если документ использует схему данных. Схема может импортироваться в таблицу стилей элементом `<xsl:import-schema>`, или документ XML может использовать атрибуты `xsi:schemaLocation` или `xsi:noNamespaceSchemaLocation`. Если с документом XML схема не связана, все узлы документа обладают типами данных и функция `nilled()` всегда возвращает `false`.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 2 «Accessors».

Пример

Очень короткий документ XML для демонстрации работы функции `nilled()`:

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

```
    xmlns="http://www.oreilly.com/xslt"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.oreilly.com/xslt person.xsd">

    <name>Doug Tidwell</name>
    <age>42</age>
    <birthday xsi:nil="true"/>
  </person>
```

Обратите внимание: в документе присутствует ссылка на определение схемы *person.xsd*. В схеме указано, что элемент `<birthday>` может иметь неопределенное значение:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- person.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name"/>
        <xs:element ref="age"/>
        <xs:element ref="birthday"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="name" type="xs:string"/>
  <xs:element name="age" type="xs:positiveInteger"/>
  <xs:element name="birthday" type="xs:date" nillable="true"/>

</xs:schema>
```

В документе XML значение элемента `<birthday>` не определено, а в схеме указано, что элемент может обладать неопределенным значением. Для демонстрации работы функции `nilled()` мы воспользуемся следующей таблицей стилей:

```
<?xml version="1.0"?>
<!-- nilled.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:po="http://www.oreilly.com/xslt">

  <xsl:output method="text"/>
  <xsl:import-schema namespace="http://www.oreilly.com/xslt"
    schema-location="person.xsd"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the nilled() function:&#xA;</xsl:text>

    <xsl:for-each select="//*">
      <xsl:text>&#xA;    Element &lt;</xsl:text>
      <xsl:value-of select="name()"/>
      <xsl:text>&gt; </xsl:text>
      <xsl:value-of select="if (nilled(.))
                            then '===>>> IS nilled!'
                            else 'is not nilled!'"/>
```

```
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
A test of the nilled() function:

    Element <person> is not nilled!
    Element <name> is not nilled!
    Element <age> is not nilled!
    Element <birthday> ===>>> IS nilled!
```

[2.0] node-name()

Возвращает расширенное полное имя для узлов, которые могут обладать именами.

Синтаксис

```
xs:QName? node-name(node()?)
```

Ввод

Узел.

Вывод

Для узлов, которым разрешено обладать именами, возвращает имя узла. Для других видов узлов (например, для узлов комментариев) `node-name()` возвращает пустую последовательность. Если аргумент `node-name()` является пустой последовательностью, функция, как и следовало ожидать, возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 2 «Accessors».

Пример

Следующая таблица стилей показывает, как функция `node-name()` работает с разными видами узлов:

```
<?xml version="1.0"?>
<!-- node-name.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Here's a test of the node-name() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:apply-templates
      select="*|comment()|processing-instruction()|text()"/>
  </xsl:template>

  <xsl:template match="*">
    <xsl:text>&#xA;</xsl:text>
    <xsl:text>  Element node: node-name(.) = '</xsl:text>
```

```
    <xsl:value-of select="node-name(.)"/>
    <xsl:if test="prefix-from-QName(node-name(.))">
      <xsl:text>'&#xA;    Prefix: '</xsl:text>
      <xsl:value-of select="prefix-from-QName(node-name(.))"/>
    </xsl:if>
    <xsl:text>'&#xA;    Local name: '</xsl:text>
    <xsl:value-of select="local-name-from-QName(node-name(.))"/>
    <xsl:if test="namespace-uri-from-QName(node-name(.))">
      <xsl:text>'&#xA;    Namespace URI: '</xsl:text>
      <xsl:value-of select="namespace-uri-from-QName(node-name(.))"/>
    </xsl:if>
    <xsl:text>'&#xA;</xsl:text>

    <xsl:if test="count(@*)">
      <xsl:for-each select="@*">
        <xsl:text>    Attribute node: node-name(.) = '</xsl:text>
        <xsl:value-of select="node-name(.)"/>
        <xsl:text>'&#xA;</xsl:text>
      </xsl:for-each>
    </xsl:if>

    <xsl:for-each select="namespace::*">
      <xsl:call-template name="namespace-node"/>
    </xsl:for-each>

    <xsl:for-each
      select="*|comment()|processing-instruction()|text()">
      <xsl:apply-templates select="."/>
    </xsl:for-each>
  </xsl:template>

  <xsl:template match="comment()">
    <xsl:text>&#xA;</xsl:text>
    <xsl:text>  Comment node: node-name(.) = '</xsl:text>
    <xsl:value-of select="node-name(.)"/>
    <xsl:text>'&#xA;</xsl:text>
  </xsl:template>

  <xsl:template match="processing-instruction()">
    <xsl:text>&#xA;</xsl:text>
    <xsl:text>  Processing instruction node: </xsl:text>
    <xsl:text>node-name(.) = '</xsl:text>
    <xsl:value-of select="node-name(.)"/>
    <xsl:text>'&#xA;</xsl:text>
  </xsl:template>

  <xsl:template match="text()">
    <xsl:if test="string-length(normalize-space(.))">
      <xsl:text>  Text node: node-name(.) = '</xsl:text>
      <xsl:value-of select="node-name(.)"/>
      <xsl:text>'&#xA;</xsl:text>
    </xsl:if>
  </xsl:template>

  <xsl:template name="namespace-node">
    <xsl:if test="string(node-name(.)) != 'xml'">
      <xsl:text>  Namespace node: node-name(.) = '</xsl:text>
      <xsl:value-of select="node-name(.)"/>
```

```
      <xsl:text>'&#xA;</xsl:text>
    </xsl:if>
  </xsl:template>

</xsl:stylesheet>
```

Для тестирования этой таблицы стилей будет использоваться слегка измененная версия документа с сонетом Шекспира (в начало документа добавлена инструкция по обработке):

```
<?xml version="1.0"?>
<!-- sonnet-pi.xml -->
<?xml-stylesheet href="node-name.xsl" type="text/xsl"?>
<!-- Слегка измененная версия знакомого документа с сонетом. -->
<sonnet type='Shakespearean'>
  <auth:author xmlns:auth="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </auth:author>
  <!-- Существует ли официальное название сонета?
       Иногда сонеты называются по первой строке. -->
  <title>Sonnet 130</title>
  <lines>
    <line>My mistress' eyes are nothing like the sun,</line>
    <line>Coral is far more red than her lips red.</line>
    <line>If snow be white, why then her breasts are dun,</line>
    <line>If hairs be wires, black wires grow on her head.</line>
    <line>I have seen roses damasked, red and white,</line>
    <line>But no such roses see I in her cheeks.</line>
    <line>And in some perfumes is there more delight</line>
    <line>Than in the breath that from my mistress reeks.</line>
    <line>I love to hear her speak, yet well I know</line>
    <line>That music hath a far more pleasing sound.</line>
    <line>I grant I never saw a goddess go,</line>
    <line>My mistress when she walks, treads on the ground.</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
  </lines>
</sonnet>
```

Наша таблица стилей универсальна, она работает с любым документом XML. Для документа с сонетом она выдает следующий результат:

```
Here's a test of the node-name() function:

  Comment node: node-name(.) = ''

  Processing instruction node: node-name(.) = 'xml-stylesheet'

  Comment node: node-name(.) = ''

  Element node: node-name(.) = 'sonnet'
    Local name: 'sonnet'
    Attribute node: node-name(.) = 'type'

  Element node: node-name(.) = 'auth:author'
    Prefix: 'auth'
```

```
  Local name: 'author'
  Namespace URI: 'http://www.authors.com/'
Namespace node: node-name(.) = 'auth'

Element node: node-name(.) = 'last-name'
  Local name: 'last-name'
Namespace node: node-name(.) = 'auth'
Text node: node-name(.) = ''

Element node: node-name(.) = 'first-name'
  Local name: 'first-name'
Namespace node: node-name(.) = 'auth'
Text node: node-name(.) = ''

...

Comment node: node-name(.) = ''

Element node: node-name(.) = 'title'
  Local name: 'title'
Text node: node-name(.) = ''

Element node: node-name(.) = 'lines'
  Local name: 'lines'

Element node: node-name(.) = 'line'
  Local name: 'line'
Text node: node-name(.) = ''

Element node: node-name(.) = 'line'
  Local name: 'line'
Text node: node-name(.) = ''
...
```

Обратите внимание: узлы элементов, атрибутов, инструкций по обработке и пространств имен обладают именами, а у комментариев и текстовых узлов имен нет. Обратите также внимание на то, что документ содержит инструкцию по обработке и узел комментария вне корневого элемента; для обращения к ним используется шаблон `match="/"`.

Функция `node-name()` возвращает значение `xs:QName`, состоящее из трех частей: префикса, локального имени и URI-адреса пространства имен. Все элементы обладают локальными именами, но, как видно из приведенных результатов, многие элементы не имеют префикса или URI-адреса пространства имен. XPath 2.0 предоставляет ряд функций для работы со значениями `xs:QName`. В этой таблице стилей функции `prefix-from-QName()`, `local-name-from-QName()` и `namespace-uri-from-QName()` используются для получения трех частей полного имени.

Наконец, инструкция по обработке `<?xml-stylesheet ...` определяет таблицу стилей по умолчанию для обработки этого документа XML. При запуске процессора XSLT обычно необходимо задать имена файла XML и таблицы стилей. Обычно процессор использует указанную таблицу стилей, игнорируя инструкцию по обработке. Чтобы процессор XSLT использовал таблицу стилей по умолчанию, обычно необходимо задать специальный параметр. (Например, для Saxon это параметр `-a`.) За подробностями обращайтесь к документации своего процессора.

normalize–space()

Удаляет лишние пропуски из строкового аргумента.

Синтаксис

```
[1.0] string normalize-space(string?)
[2.0] xs:string normalize-space(xs:string?)
```

Ввод

Необязательная строка. Если аргумент не указан, функция `normalize-space()` использует строковое значение контекстного узла.

[2.0] Если аргумент является пустой последовательностью, `normalize-space()` возвращает строку нулевой длины.

Вывод

Исходная строка, из которой удалены пропуски по следующим правилам:

- Все начальные пропуски удаляются.
- Все конечные пропуски удаляются.
- Внутри строки каждая серия пропусков заменяется одним пробелом.

Если строка состоит из одних пропусков, функция возвращает строку нулевой длины.

[2.0] В XSLT 2.0 вызов `normalize-space()` для пустой последовательности возвращает пустую последовательность.

Где определяется

[1.0] XPath, раздел 4.2 «String Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.4 «Functions on String Values».

Пример

Короткий пример, демонстрирующий работу функции `normalize-space()`:

```
<?xml version="1.0"?>
<!-- normalize-space1.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:variable name="newline">
    <xsl:text>&#xA;</xsl:text>
  </xsl:variable>

  <xsl:variable name="testString">
    <xsl:text>                 This
is

a string
that had        &#x9;&#x9;    lots of

&#xA;&#xA;&#xA;
whitespace.
```

```
</xsl:text>
    </xsl:variable>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the normalize-space() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  normalize-space('       </xsl:text>
    <xsl:text>Hello,            World!')="</xsl:text>
    <xsl:value-of
      select="normalize-space('       Hello,            World!')"/>
    <xsl:text>"</xsl:text>
    <xsl:text>&#xA;  normalize-space($newline)="</xsl:text>
    <xsl:value-of select="normalize-space($newline)"/>
    <xsl:text>"&#xA;</xsl:text>
    <xsl:text>  normalize-space($testString)=&#xA;    "</xsl:text>
    <xsl:value-of select="normalize-space($testString)"/>
    <xsl:text>"</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Tests of the normalize-space() function:

  normalize-space('       Hello,            World!')="Hello, World!"
  normalize-space($newline)=""
  normalize-space($testString)=
    "This is a string that had lots of whitespace."
```

В первом и третьем тестах `normalize-space()` **серии пробелов, табуляций (**`	`**) и символов новой строки (**`
`**) в строке заменяются одиночными пробелами. Во втором тесте переменной присваивается значение** `
`**. Это строка, состоящая из одних пропусков, поэтому нормализованная версия такой строки представляет собой строку нулевой длины.**

Другая таблица стилей тестирует поведение XSLT 2.0 при вызове `normalize-space()` **с пустой последовательностью на входе:**

```
<?xml version="1.0"?>
<!-- normalize-space2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the normalize-space() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  normalize-space(())="</xsl:text>
    <xsl:value-of select="normalize-space(())"/>
    <xsl:text>"</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Результат вполне предсказуем:

```
A test of the normalize-space() function:

  normalize-space(())=""
```

[2.0] normalize-unicode()

Для заданной строки функция возвращает строку с символами, преобразованными к конкретной нормализованной форме Юникода.

Синтаксис

```
xs:string normalize-unicode(xs:string?)
xs:string normalize-unicode(xs:string?, $normalizationForm as xs:string)
```

Ввод

Нормализуемая строка. При вызове может передаваться необязательная строка с именем формы нормализации. Допустимые формы нормализации Юникода: `NFC`, `NFD`, `NFKC`, `NFKD` и `FULLY-NORMALIZED`.

В общем случае формы нормализации Юникода определяют способ обработки символов. Например, символ `A` может быть представлен как одним символом (`Å`), так и прописной буквой A (`Å`), за которой следует комбинационный символ «верхнего кольца» (`̊`). Эти два представления семантически эквивалентны, но различаются на символьном уровне. Механизм нормализации преобразует разные представления в единую форму для сравнения.

Все процессоры, построенные в соответствии со спецификациями, должны поддерживать формат `NFC`. Кроме того, они могут поддерживать любые из форматов, определяемых в Юникоде, а также (по желанию разработчиков) свои собственные нестандартные форматы.

При желании узнать больше обращайтесь к документу Unicode Standard Annex #15, Unicode Normalization Forms по адресу *http://www.unicode.org/unicode/reports/tr15/*.

Вывод

Строка, полученная преобразованием символов входной строки в соответствующую форму нормализации Юникода. Если значение первого аргумента является пустой последовательностью, `normalize-unicode()` возвращает строку нулевой длины.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.4 «Functions on String Values».

Пример

Очень короткий пример со сравнением двух только что упоминавшихся строк:

```
<?xml version="1.0"?>
<!-- normalize-unicode.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Here is a test of the normalize-</xsl:text>
    <xsl:text>unicode() function:&#xA;</xsl:text>

    <xsl:text>&#xA;  compare('&#xC5', </xsl:text>
    <xsl:text>'&#x41;&#x30A;') = </xsl:text>
```

```
    <xsl:value-of select="compare('&#xC5;', '&#x41;&#x30A;')"/>

    <xsl:text>&#xA;&#xA;  compare(normalize-unicode</xsl:text>
    <xsl:text>('&#xC5'),&#xA;           normalize-unicode(</xsl:text>
    <xsl:text>'&#x41;&#x30A;')) = </xsl:text>
    <xsl:value-of
      select="compare(normalize-unicode('&#xC5;'),
              normalize-unicode('&#x41;&#x30A;'))"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here is a test of the normalize-unicode() function:

  compare('&#xC5', '&#x41;&#x30A;') = 1

  compare(normalize-unicode('&#xC5'),
          normalize-unicode('&#x41;&#x30A;')) = 0
```

При первом сравнении две строки не равны, а при втором сравнении нормализованные версии строк равны.

not()

Возвращает результат логического отрицания аргумента. Если аргумент не является логическим значением, он преобразуется в логическое значение по правилам, представленным в описании функции `boolean()`.

Синтаксис

```
[1.0] boolean not(boolean)
[2.0] xs:boolean not(item()*)
```

Ввод

Логическое значение или чаще выражение XPath с логическим результатом.

Вывод

`false`, если входной параметр имеет истинное значение; `true`, если входной параметр имеет ложное значение.

Где определяется

[1.0] XPath, раздел 4.3 «Boolean Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 9.3 «Functions on Boolean Values».

Пример

Для демонстрации работы функции `not()` будут использоваться те же таблица стилей и документ XML, что и для функции `boolean()`. Документ XML:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
```

```
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Документ обрабатывается по следующей таблице стилей, в которой вместо `boolean()` используется `not()`. Функция `boolean()` преобразует свой аргумент в логическое значение; функция `not()` неявно использует `boolean()` для преобразования аргумента в логическую величину, а затем инвертирует его.

```
<?xml version="1.0"?>
<!-- not.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the not() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  not((true()) = </xsl:text>
    <xsl:value-of select="not(true())"/>

    <xsl:text>&#xA;  not(true) = </xsl:text>
    <xsl:value-of select="not(true)"/>

    <xsl:text>&#xA;  not('false') = </xsl:text>
    <xsl:value-of select="not('false')"/>

    <xsl:text>&#xA;  not('7') = </xsl:text>
    <xsl:value-of select="not('7')"/>

    <xsl:text>&#xA;  not(/report/brand/units[. > 20000]) = </xsl:text>
    <xsl:value-of select="not(/report/brand/units[. > 20000])"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the not() function:

  not((true()) = false
  not(true) = true
  not('false') = false
```

```
not('7') = false
not(/report/brand/units[. > 20000]) = false
```

Как и следовало ожидать, результаты полностью противоположны тем, которые были получены при тестировании функции `boolean()`. В первом тесте в аргументе передается вызов функции `true()`, который всегда возвращает истинное значение. Отрицанием логической истины является логическая ложь (`false`). Во втором тесте запрашиваются все элементы `<true>` из текущего контекста; таких элементов нет, поэтому результат равен `false`, а его отрицание дает `true`. В аргументе третьего теста передается строка, длина которой больше нуля, поэтому `not()` возвращает `false` (строки ненулевой длины интерпретируются как логическая истина). Аналогично аргумент четвертого теста является строкой положительной длины, поэтому `not()` возвращает `false` и в этом случае.

В последнем примере команда XPath выбирает из отчета все элементы `<units>` с числовым значением больше 20000. Так как результат содержит как минимум один узел, выражение интерпретируется как истинное, а `not()` возвращает `false`. Такое решение работает в XSLT 1.0, потому что оно генерирует набор узлов, содержащий хотя бы один узел; оно также работает в XSLT 2.0, потому что оно создает последовательность, первым объектом которой является узел. Если изменить выражение так, чтобы оно искало информацию о продажах на сумму более 30000, `not()` вернет `true`.

number()

Преобразует аргумент в число.

Синтаксис

```
[1.0] number number(object?)
[2.0] xs:double number()
[2.0] xs:double number(xs:anyAtomicType)
```

Ввод

[1.0] Объект.

[2.0] Значение `xs:anyAtomicType`.

Вывод

Число. Если функция вызвана без аргумента, используется контекстный объект. В XSLT 1.0 и 2.0 используются разные правила преобразования аргумента в число.

[1.0] Правила для XSLT 1.0:

- Если аргумент является логическим значением, `true` преобразуется в число `1`, а `false` преобразуется в число `0`.
- Если аргумент является набором узлов, он преобразуется в строку так, как если бы он был передан функции `string()`, после чего строка преобразуется в число по обычным правилам. (Напомню, что функция `string()` возвращает строковое значение первого узла в наборе узлов.)
- Если аргумент является строкой, преобразование выполняется по следующим правилам:
  - Если строка состоит из необязательных пропусков, за которыми следует необязательный знак «минус» (-) с последующим числом и пропусками, она преобразуется в вещественное значение, ближайшее к математическому значе-

нию, представляемому строкой. (Правила округления определяются в стандарте IEEE 754; за дополнительной информацией обращайтесь к стандарту.)

- Любая другая строка преобразуется в значение `NaN` («нечисло»).

- Процессоры XSLT 1.0 могут поддерживать другие типы данных. Если аргумент относится к другому типу, он преобразуется в число способом, зависящим от типа. За информацией о том, какие типы поддерживаются и как они преобразуются в числа, обращайтесь к документации своего процессора XSLT.

[2.0] XSLT 2.0 поддерживает большее количество типов данных, поэтому правила усложняются:

- Если аргумент уже относится к типу `xs:double`, он возвращается в неизменном виде.
- Если аргумент относится к типу `xs:boolean`, `true` возвращается в виде `1.0E0`, а `false` — в виде `0.0E0`.
- Если аргумент относится к типу `xs:float`, `xs:decimal` или `xs:integer`, он преобразуется к типу `xs:double` и возвращается функцией. Если аргумент относится к типу `xs:float` и является одним из `xs:float` значений `INF`, `-INF`, `NaN`, положительный нуль или отрицательный нуль, он возвращается в виде этих значений для типа `xs:double`.
- Если аргумент является пустой последовательностью, возвращается `NaN` («нечисло»).
- Если аргумент (или контекстный узел при отсутствии аргумента) не может быть преобразован к типу `xs:double`, возвращается значение `NaN`. (Например, типы данных `xs:date`, `xs:dateTime` и `xs:time` не могут быть преобразованы в числа.)

Где определяется

[1.0] XPath, раздел 4.4 «Number Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 14 «Functions and Operators on Nodes».

Пример

Для демонстрации работы функции `number()` будет использоваться документ с информацией о продажах шоколада:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
```

```
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Функция `number()` **тестируется с различными аргументами:**

```
<?xml version="1.0"?>
<!-- number.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the number() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  number(true()) = </xsl:text>
    <xsl:value-of select="number(true())"/>
    <xsl:text>&#xA;&#xA;  number(false()) = </xsl:text>
    <xsl:value-of select="number(false())"/>
    <xsl:text>&#xA;&#xA;  number(/report/brand[2]/units) = </xsl:text>
    <xsl:value-of select="number(/report/brand[2]/units)"/>
    <xsl:text>&#xA;&#xA;  number(//units) = </xsl:text>
    <xsl:value-of select="number(//units)"/>
    <xsl:text>&#xA;&#xA;  number(/report/title) = </xsl:text>
    <xsl:value-of select="number(/report/title)"/>
  </xsl:template>

</xsl:stylesheet>
```

Вывод таблицы стилей выглядит так:

```
Tests of the number() function:

  number(true()) = 1

  number(false()) = 0

  number(/report/brand[2]/units) = 8203

  number(//units) = 27408

  number(/report/title) = NaN
```

Это таблица стилей XSLT 1.0, поэтому выражение `number(//units)` **будет выполняться в процессоре XSLT 2.0, работающем в режиме совместимости с XSLT 1.0. В XSLT 2.0 процессор выдает ошибку при передаче нескольких узлов функции** `number()`**. Слегка измененная таблица стилей для XSLT 2.0:**

```
<?xml version="1.0"?>
<!-- number2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the number() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  number(true()) = </xsl:text>
```

```
    <xsl:value-of select="number(true())"/>
    <xsl:text>&#xA;&#xA;  number(false()) = </xsl:text>
    <xsl:value-of select="number(false())"/>
    <xsl:text>&#xA;&#xA;  number(/report/brand[2]/units) = </xsl:text>
    <xsl:value-of select="number(/report/brand[2]/units)"/>
    <xsl:text>&#xA;&#xA;  number((//units)[1]) = </xsl:text>
    <xsl:value-of select="number((//units)[1])"/>
    <xsl:text>&#xA;&#xA;  number(/report/title) = </xsl:text>
    <xsl:value-of select="number(/report/title)"/>
    <xsl:text>&#xA;&#xA;  number(current-date()) = </xsl:text>
    <xsl:value-of select="number(current-date())"/>
    <xsl:text>&#xA;&#xA;  number(current-time()) = </xsl:text>
    <xsl:value-of select="number(current-time())"/>
  </xsl:template>

</xsl:stylesheet>
```

Обратите внимание: нам пришлось заменить вызов `number(//units)` на `number((//units)[1])`. Он получает все элементы `<units>` на всех уровнях документа и выбирает первый из них. В любом случае результат выглядит так:

```
Tests of the number() function:

  number(true()) = 1

  number(false()) = 0

  number(/report/brand[2]/units) = 8203

  number((//units)[1]) = 27408

  number(/report/title) = NaN

  number(current-date()) = NaN

  number(current-time()) = NaN
```

Обратите внимание: значения `xs:date` и `xs:time` возвращаются как `NaN`.

[2.0] one-or-more()

Выдает ошибку, если аргумент не является последовательностью, содержащей один или более объектов. Функция может использоваться и для проверки типов. Допустим, элемент `<addressbook>` обязательно должен содержать хотя бы один элемент `<address>`. Мы можем вызвать `one-or-more(address)` для текущего элемента `<addressbook>`; если он не содержит ни одного элемента `<address>`, процессор XSLT выдает ошибку.

Синтаксис

```
item()+ one-or-more(item()*)
```

Ввод

Последовательность. Если последовательность не содержит по крайней мере одного объекта, `one-or-more()` выдает ошибку.

Вывод

Если входная последовательность содержит хотя бы один объект, `one-or-more()` возвращает эту последовательность. Если входная последовательность является пустой последовательностью, `one-or-more()` выдает ошибку.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.2 «Functions That Test the Cardinality of Sequences». Дополнительная информация об этой функции содержится также в XQuery 1.0 and XPath 2.0 Formal Semantics, раздел 7.2 «Standard Functions with Specific Static Typing Rules».

Пример

Простая таблица стилей для демонстрации работы функции `one-or-more()`:

```
<?xml version="1.0"?>
<!-- one-or-more.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <xsl:variable name="testSequence" as="item()*">
      <xsl:sequence
        select="(3, 4, 5, current-date(), current-time(), 8, 'blue',
                'red', xs:float(3.14), 42, xs:date('1995-04-21'))"/>
    </xsl:variable>

    <xsl:text>&#xA;Here is a test of the one-or-more() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  Our sequence is:&#xA;&#xA;    </xsl:text>
    <xsl:value-of select="$testSequence" separator="&#xA;    "/>

    <xsl:if test="count(one-or-more($testSequence))">
      <xsl:text>&#xA;&#xA;  Our test sequence has one or </xsl:text>
      <xsl:text>more items!</xsl:text>
    </xsl:if>

  </xsl:template>

</xsl:stylesheet>
```

Результат применения таблицы стилей:

```
Here is a test of the one-or-more() function:

  Our sequence is:

    3
    4
    5
    2006-07-22-04:00
    23:04:38.183-04:00
    8
    blue
    red
    3.14
    42
    1995-04-21

  Our test sequence has one or more items!
```

Обратите внимание на использование функции count() в атрибуте test элемента <xsl:if>. На первый взгляд следующая запись элемента <xsl:if> выглядит вполне разумно:

```
<xsl:if test="one-or-more($testSequence)">  <!-- Не работает! -->
```

Функция one-or-more() возвращает входную последовательность при условии, что она содержит хотя бы одно значение. Процессор XSLT пытается преобразовать содержимое атрибута test к логическому типу. Если последовательность, возвращаемая one-or-more(), состоит из одного объекта *или* ее первый объект является узлом, все работает. Однако с последовательностью из нашего примера процессор XSLT выдает ошибку. Последовательность из нескольких объектов, первый объект которой является атомарным значением, не может быть преобразована в логическое значение. Вот почему элемент <xsl:if> приходится записывать в следующем виде:

```
<xsl:if test="count(one-or-more($testSequence))"/>
```

Процессор XSLT преобразует числовое значение, возвращаемое count(), в логическое значение. Это значение всегда не менее 1, поэтому атрибут test вычисляется как true при каждом выполнении функции one-or-more().

См. также

Описания функций *[2.0]* exactly-one() и *[2.0]* zero-or-one().

position()

Возвращает позицию контекстного объекта в последовательности, обрабатываемой в настоящий момент.

Синтаксис

```
[1.0] number position()
[2.0] xs:integer position()
```

Ввод

Нет.

Вывод

Число, равное позиции текущего узла в последовательности, обрабатываемой в настоящий момент. Например, если текущий узел является пятым обрабатываемым элементом <li>, вызов position() вернет 5.

Где определяется

[1.0] XPath, раздел 4.1 «Node Set Functions»,

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 16 «Context Functions».

Примеры

В этом примере функция position() используется для создания атрибута style для ячеек таблицы HTML. Цвет фона переключается между black (черным), gray (серым) и white (белым), а основной цвет – между white (белым), white (белым) и black (черным). Мы будем использовать следующий документ XML:

```
<?xml version="1.0"?>
<!-- albums.xml -->
<list xml:lang="en">
  <title>Albums I've bought recently:</title>
  <listitem>The Sacred Art of Dub</listitem>
  <listitem>Only the Poor Man Feel It</listitem>
  <listitem>Excitable Boy</listitem>
  <listitem xml:lang="sw">Aki Special</listitem>
  <listitem xml:lang="en-gb">Combat Rock</listitem>
  <listitem xml:lang="zu">Talking Timbuktu</listitem>
  <listitem xml:lang="jz">The Birth of the Cool</listitem>
</list>
```

Для построения документа HTML будет использоваться следующая таблица стилей:

```
<?xml version="1.0"?>
<!-- position.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>
          <xsl:value-of select="/list/title"/>
        </title>
      </head>
      <body style="font-family: sans-serif;">
        <h1>
          <xsl:value-of select="/list/title"/>
        </h1>
        <table border="1" cellpadding="5">
          <xsl:for-each select="/list/listitem">
            <xsl:variable name="style">
              <xsl:text>color: </xsl:text>
              <xsl:choose>
                <xsl:when test="position() mod 3 = 1">white</xsl:when>
                <xsl:when test="position() mod 3 = 2">white</xsl:when>
                <xsl:otherwise>black</xsl:otherwise>
              </xsl:choose>
              <xsl:text>; background: </xsl:text>
              <xsl:choose>
                <xsl:when test="position() mod 3 = 1">black</xsl:when>
                <xsl:when test="position() mod 3 = 2">gray</xsl:when>
                <xsl:otherwise>white</xsl:otherwise>
              </xsl:choose>
              <xsl:text>;</xsl:text>
            </xsl:variable>
            <tr style="{$style}">
              <td>
                <b><xsl:value-of select="."/></b>
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
```

```
    </xsl:template>
</xsl:stylesheet>
```

Функция `position()` используется для перебора цветов фона и текста. Таблица стилей выдает следующий результат:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Albums I've bought recently:</title>
   </head>
   <body style="font-family: sans-serif;">
      <h1>Albums I've bought recently:</h1>
      <table border="1" cellpadding="5">
         <tr style="color: white; background: black;">
            <td><b>The Sacred Art of Dub</b></td>
         </tr>
         <tr style="color: white; background: gray;">
            <td><b>Only the Poor Man Feel It</b></td>
         </tr>
         <tr style="color: black; background: white;">
            <td><b>Excitable Boy</b></td>
         </tr>
         <tr style="color: white; background: black;">
            <td><b>Aki Special</b></td>
         </tr>
         <tr style="color: white; background: gray;">
            <td><b>Combat Rock</b></td>
         </tr>
         <tr style="color: black; background: white;">
            <td><b>Talking Timbuktu</b></td>
         </tr>
         <tr style="color: white; background: black;">
            <td><b>The Birth of the Cool</b></td>
         </tr>
      </table>
   </body>
</html>
```

На рис. С.5 показано, как выглядит файл HTML в броузере.

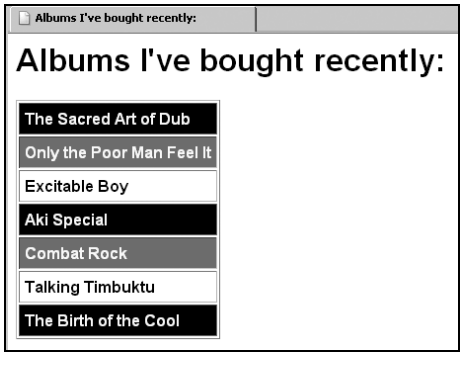


***Рис. С.5.** Файл HTML с чередованием цветов фона и текста*

[2.0] prefix-from-QName()

Для заданного значения xs:QName возвращает префикс пространства имен.

Синтаксис

```
xs:NCName? prefix-from-QName(xs:QName?)
```

Ввод

Значение xs:QName.

Вывод

Значение xs:NCName (имя «без двоеточия»), представляющее префикс из полного имени. Если полное имя не имеет префикса или аргумент xs:QName является пустой последовательностью, prefix-from-QName() возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 11.2 «Operators and Functions Related to QNames».

Пример

Как и в других примерах работы с пространствами имен, в качестве входного документа XML мы используем сонет Шекспира:

```
<?xml version="1.0"?>
<!-- sonnet.xml -->
<sonnet type='Shakespearean'>
  <auth:author xmlns:auth="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </auth:author>
  <!-- Существует ли официальное название сонета?
       Иногда сонеты называются по первой строке. -->
  <title>Sonnet 130</title>
  <lines>
    <line>My mistress' eyes are nothing like the sun,</line>
    <line>Coral is far more red than her lips red.</line>
    <line>If snow be white, why then her breasts are dun,</line>
    <line>If hairs be wires, black wires grow on her head.</line>
    <line>I have seen roses damasked, red and white,</line>
    <line>But no such roses see I in her cheeks.</line>
    <line>And in some perfumes is there more delight</line>
    <line>Than in the breath that from my mistress reeks.</line>
    <line>I love to hear her speak, yet well I know</line>
    <line>That music hath a far more pleasing sound.</line>
    <line>I grant I never saw a goddess go,</line>
    <line>My mistress when she walks, treads on the ground.</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
  </lines>
</sonnet>
```

Таблица стилей:

```
<?xml version="1.0"?>
<!-- prefix-from-qname.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the prefix-from-QName() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:variable name="testQName1" as="xs:QName"
      select="resolve-QName('auth:last-name',
              /sonnet/*:author/last-name)"/>
    <xsl:text>&#xA;  testQName1 = resolve-QName('auth:last-name', </xsl:text>
    <xsl:text>/sonnet/*:author/last-name)</xsl:text>
    <xsl:text>&#xA;  prefix-from-QName($testQName1) = "</xsl:text>
    <xsl:value-of select="prefix-from-QName($testQName1)"/>
    <xsl:text>"</xsl:text>

    <xsl:variable name="testQName2" as="xs:QName"
      select="resolve-QName('something-else', /sonnet)"/>
    <xsl:text>&#xA;&#xA;  testQName2 = resolve-QName('</xsl:text>
    <xsl:text>something-else', /sonnet)</xsl:text>
    <xsl:text>&#xA;  prefix-from-QName($testQName2) = "</xsl:text>
    <xsl:value-of select="prefix-from-QName($testQName2)"/>
    <xsl:text>"</xsl:text>

    <xsl:variable name="testQName3" as="xs:QName"
      select="QName('http://www.authors.com/', 'pfx:writer')"/>
    <xsl:text>&#xA;&#xA;  testQName3 = QName('http://www.</xsl:text>
    <xsl:text>authors.com/', 'pfx:writer')&#xA;</xsl:text>
    <xsl:text>  prefix-from-QName($testQName3) = "</xsl:text>
    <xsl:value-of select="prefix-from-QName($testQName3)"/>
    <xsl:text>"</xsl:text>

  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the prefix-from-QName() function:

  testQName1 = resolve-QName('auth:last-name', /sonnet/*:author/last-name)
  prefix-from-QName($testQName1) = "auth"

  testQName2 = resolve-QName('something-else', /sonnet)
  prefix-from-QName($testQName2) = ""

  testQName3 = QName('http://www.authors.com/', 'pfx:writer')
  prefix-from-QName($testQName3) = "pfx"
```

Аргумент функции `prefix-from-QName()` ***должен*** **относиться к типу** `xs:QName`**. В приведенной таблице стилей полные имена конструируются функцией** `resolve-QName()` **и функцией-конструктором** `QName()`**.**

В первом тесте для создания `QName` **используется полное имя элемента** `auth:lastname`**. Контекстом разрешения полного имени (второй аргумент** `resolve-QName()`**) является**

элемент <last-name> нашего сонета. Функция resolve-QName() возвращает полное имя с локальным именем last-name, префиксом пространства имен auth и URI-адресом пространства имен http://www.authors.com/. Если ни один узел не соответствует выражению XPath /sonnet/*:author/last-name, вызов resolve-QName() завершается неудачей. Сбой происходит и в том случае, если для элемента <last-name> не определен префикс auth.

Во втором тесте prefix-from-QName() при вызове resolve-QName() используется имя элемента без префикса. Документ XML не имеет пространства имен по умолчанию, поэтому в полном имени префикс и URI-адрес пространства имен представлены пустыми строками.

В последнем тесте функция QName() создает полное имя «с нуля». В двух аргументах функции передаются URI-адрес пространства имен и имя элемента – в данном случае с префиксом. Полученное полное имя образуется из локального имени writer, префикса пространства имен pfx и URI-адреса пространства имен http://www.authors.com/.

[2.0] QName()

Функция-конструктор для создания новых значений QName.

Синтаксис

```
xs:QName QName($paramURI as xs:string?, $paramQName as xs:string)
```

Ввод

Два значения xs:string. Первая строка определяет URI-адрес пространства имен полного имени, а вторая – строковое значение полного имени. Если вторая строка содержит двоеточие (:), подстрока перед двоеточием интерпретируется как префикс пространства имен полного имени, а подстрока после двоеточия – как локальное имя. Например, если во втором аргументе передается строка auth:author, то она определяет префикс пространства имен auth и локальное имя author.

URI-адрес пространства имен может быть строкой нулевой длины или пустой последовательностью; это означает, что полное имя не принадлежит к пространству имен. В этом случае процессор XSLT выдает ошибку, если вторая строка содержит двоеточие. (Другими словами, при отсутствии URI-адреса пространства имен нельзя иметь префикс пространства имен.)

Вывод

Значение xs:QName.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 11.1 «Additional Constructor Functions for QNames».

Пример

Следующая таблица стилей создает несколько полных имен, а затем при помощи различных функций выводит дополнительную информацию о них:

```
<?xml version="1.0"?>
<!-- qname.xsl -->
<xsl:stylesheet version="2.0"
```

```
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xsl:output method="text"/>

    <xsl:template match="/">
      <xsl:text>&#xA;Tests of the QName() constructor function:&#xA;</xsl:text>

      <xsl:variable name="testQName1" as="xs:QName"
        select="QName('http://www.authors.com', 'auth:author')"/>

      <xsl:text>&#xA;  testQName1 = QName('http://www.</xsl:text>
      <xsl:text>authors.com', 'auth:author')</xsl:text>
      <xsl:text>&#xA;  prefix-from-QName</xsl:text>
      <xsl:text>($testQName1) = "</xsl:text>
      <xsl:value-of select="prefix-from-QName($testQName1)"/>
      <xsl:text>"&#xA;  local-name-from-QName($testQName1) = "</xsl:text>
      <xsl:value-of select="local-name-from-QName($testQName1)"/>
      <xsl:text>"&#xA;  namespace-uri-from-QName($testQName1) = "</xsl:text>
      <xsl:value-of select="namespace-uri-from-QName($testQName1)"/>
      <xsl:text>"</xsl:text>
      <xsl:variable name="testQName2" as="xs:QName"
        select="QName('', 'sonnet')"/>

      <xsl:text>&#xA;&#xA;  testQName2 = QName('', 'sonnet')</xsl:text>
      <xsl:text>&#xA;  prefix-from-QName</xsl:text>
      <xsl:text>($testQName2) = "</xsl:text>
      <xsl:value-of select="prefix-from-QName($testQName2)"/>
      <xsl:text>"&#xA;  local-name-from-QName($testQName2) = "</xsl:text>
      <xsl:value-of select="local-name-from-QName($testQName2)"/>
      <xsl:text>"&#xA;  namespace-uri-from-QName($testQName2) = "</xsl:text>
      <xsl:value-of select="namespace-uri-from-QName($testQName2)"/>
      <xsl:text>"</xsl:text>

      <xsl:variable name="testQName3" as="xs:QName"
        select="QName('http://www.authors.com/', 'writer')"/>

      <xsl:text>&#xA;&#xA;  testQName3 = QName('http://www.</xsl:text>
      <xsl:text>authors.com/', 'writer')</xsl:text>
      <xsl:text>&#xA;  prefix-from-QName</xsl:text>
      <xsl:text>($testQName3) = "</xsl:text>
      <xsl:value-of select="prefix-from-QName($testQName3)"/>
      <xsl:text>"&#xA;  local-name-from-QName($testQName3) = "</xsl:text>
      <xsl:value-of select="local-name-from-QName($testQName3)"/>
      <xsl:text>"&#xA;  namespace-uri-from-QName($testQName3) = "</xsl:text>
      <xsl:value-of select="namespace-uri-from-QName($testQName3)"/>
      <xsl:text>"</xsl:text>

    </xsl:template>

  </xsl:stylesheet>
```

Результат:

```
  Tests of the QName() constructor function:

    testQName1 = QName('http://www.authors.com', 'auth:author')
    prefix-from-QName($testQName1) = "auth"
    local-name-from-QName($testQName1) = "author"
    namespace-uri-from-QName($testQName1) = "http://www.authors.com"

    testQName2 = QName('', 'sonnet')
```

```
prefix-from-QName($testQName2) = ""
local-name-from-QName($testQName2) = "sonnet"
namespace-uri-from-QName($testQName2) = ""

testQName3 = QName('http://www.authors.com/', 'writer')
prefix-from-QName($testQName3) = ""
local-name-from-QName($testQName3) = "writer"
namespace-uri-from-QName($testQName3) = "http://www.authors.com/"
```

Таблица стилей создает три полных имени, а затем для каждого из них получает префикс пространства имен, локальное имя и URI-адрес пространства имен. В первом тесте создается полное имя с префиксным именем элемента `auth:last-name`. Во втором тесте используются пустое пространство имен и имя элемента без префикса, а в третьем – URI-адрес пространства имен без префикса элемента.

[2.0] regex-group()

При использовании регулярного выражения для обработки строки эта функция возвращает часть анализируемой строки, соответствующую одной из групп регулярного выражения. Функция используется только внутри элемента `<xsl:matching-substring>`.

Синтаксис

```
xs:string regex-group(xs:integer)
```

Ввод

Значение `xs:integer`, представляющее частичное совпадение регулярного выражения.

Вывод

Часть анализируемой строки, совпадающая с заданной частью регулярного выражения. Чтобы часть регулярного выражения считалась группой, она должна быть заключена в круглые скобки.

Несколько замечаний по поводу функции `regex-group()`:

- Вызов `regex-group(0)` возвращает всю совпавшую подстроку, включая символы, не входящие ни в одну группу.
- Если указанная часть регулярного выражения существует, но не имеет совпадения в анализируемой строке, `regex-group()` возвращает строку нулевой длины. (Иначе говоря, если группа описывает необязательную часть совпадения.)
- Если указанная часть регулярного выражения существует, но имеет совпадение нулевой длины в анализируемой строке, `regex-group()` возвращает строку нулевой длины.
- Если указанная часть регулярного выражения не существует (например, при вызове `regex-group(4)`, когда регулярное выражение содержит только три группы), `regex-group()` возвращает строку нулевой длины.
- Наконец, если номер группы является отрицательным числом, `regex-group()` возвращает строку нулевой длины.

Где определяется

XSLT 2.0, раздел 15 «Regular Expressions».

Пример

Мы воспользуемся упрощенной версией одной из таблиц стилей, которая применялась ранее для демонстрации возможностей элемента `<xsl:analyze-string>`:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- regex-group.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Formatted phone numbers:&#xA;</xsl:text>
    <xsl:for-each select="phonelist/phonenumber">
      <xsl:analyze-string select="."
       regex="([0-9]{{3}})-([0-9]{{3}})-([0-9]{{4}})">
       <xsl:matching-substring>
         <xsl:text>&#xA;  The data </xsl:text>
         <xsl:value-of select="regex-group(0)"/>
         <xsl:text> matches.  </xsl:text>
         <xsl:text>&#xA;    The formatted phone number is: +1 (</xsl:text>
         <xsl:value-of select="regex-group(1)"/>
         <xsl:text>) </xsl:text>
         <xsl:value-of select="regex-group(2)"/>
         <xsl:text>-</xsl:text>
         <xsl:value-of select="regex-group(3)"/>
       </xsl:matching-substring>
      </xsl:analyze-string>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Регулярное выражение используется для анализа телефонных номеров. Выражение `([0-9]{{3}})-([0-9]{{3}})-([0-9]{{4}})` содержит три группы: `([0-9]{{3}})`, `([0-9]{{3}})` и `([0-9]{{4}})`. Два дефиса между группами не заключены в круглые скобки, поэтому они не принадлежат ни к одной группе.

Приведенная таблица стилей будет использоваться для обработки очень простого документа:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- simple-phonelist.xml -->
<phonelist>
  <phonenumber>Doug's number is 919-555-1212</phonenumber>
  <phonenumber>617-555-1212</phonenumber>
</phonelist>
```

Результат выглядит так:

```
Formatted phone numbers:

  The data 919-555-1212 matches.
    The formatted phone number is: +1 (919) 555-1212
  The data 617-555-1212 matches.
    The formatted phone number is: +1 (617) 555-1212
```

Для каждого совпадения `regex-group(0)` возвращает всю совпавшую строку, `regex-group(1)` возвращает код зоны (в примере используются телефонные номера США

и Канады), regex-group(2) возвращает код телефонной станции, а regex-group(3) – последние четыре цифры телефонного номера. Обратите внимание: regex-group(0) возвращает всю строку совпадения, включая два дефиса, не принадлежащие ни к одной из групп. В первом примере исходного файла XML regex-group(0) *не* возвращает символы, находящиеся за пределами совпадения, – только часть строки, совпавшую с регулярным выражением.

Наконец, если бы регулярное выражение допускало двух- или трехбуквенное сокращенное обозначение штата или провинции в конце телефонного номера (при использовании регулярного выражения ([0-9]{{3}})-([0-9]{{3}})-([0-9]{{4}})([A-Z]{{2,3}})?), вызов regex-group(4) вернул бы строку нулевой длины для обоих элементов <phonenumber> из исходного документа. Хотя формально совпадение существует, оно не содержит данных. В этом случае для проверки наличия данных в совпадении можно было бы воспользоваться конструкцией string-length(regex-group(4)).

См. также

Определение элемента *[2.0]* <xsl:analyze-string>.

[2.0] remove()

Удаляет элемент, находящийся в заданной позиции указанной последовательности.

Синтаксис

```
item()* remove($target as item()*, $position as xs:integer)
```

Ввод

Последовательность и xs:integer.

Вывод

Функция возвращает входную последовательность, из которой был удален элемент в заданной позиции. Если позиция меньше 1 или больше длины последовательности, последовательность возвращается без изменений. Наконец, для пустой последовательности возвращается пустая последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.1 «General Functions and Operators on Sequences».

Пример

Мы рассмотрим три таблицы стилей, демонстрирующие работу функции remove(). В каждой таблице стилей используется свой тип последовательности. Первая таблица стилей использует синглетную последовательность:

```
<?xml version="1.0"?>
<!-- remove1.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
```

```
    <xsl:variable name="singleton" as="item()*">
      <xsl:sequence select="(3)"/>
    </xsl:variable>

    <xsl:text>&#xA;Here is a test of the remove() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  Our sequence ($singleton) </xsl:text>
    <xsl:text>is:&#xA;      </xsl:text>
    <xsl:value-of select="$singleton" separator="&#xA;    "/>
    <xsl:text>&#xA;&#xA;    remove($singleton, 0) = </xsl:text>
    <xsl:value-of select="remove($singleton, 0)"
      separator="&#xA;    "/>
    <xsl:text>&#xA;&#xA;    remove($singleton, 7) = </xsl:text>
    <xsl:value-of select="remove($singleton, 7)"
      separator="&#xA;    "/>
    <xsl:text>&#xA;&#xA;    remove($singleton, 1) = </xsl:text>
    <xsl:value-of select="remove($singleton, 1)"
      separator="&#xA;    "/>
    <xsl:text>&#xA;&#xA;    empty(remove($singleton, 1)) = </xsl:text>
    <xsl:value-of select="empty(remove($singleton, 1))"/>

  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here is a test of the remove() function:

  Our sequence ($singleton) is:
      3

    remove($singleton, 0) = 3

    remove($singleton, 7) = 3

    remove($singleton, 1) =

    empty(remove($singleton, 1)) = true
```

Удаление объектов в позициях `0` и `7` не изменяет исходной последовательности. При удалении первого объекта из синглетной последовательности возвращается пустая последовательность, как и следовало ожидать. В последнем тесте функция `empty()` демонстрирует, что при удалении одного объекта из синглетной последовательности действительно получается пустая последовательность.

В следующем тесте функции `remove()` используется более длинная последовательность, содержащая разные типы данных:

```
<?xml version="1.0"?>
<!-- remove2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <xsl:variable name="longSequence" as="item()*">
      <xsl:sequence
```

```
        select="(3, 4, 5, current-date(), current-time(), 8, 'blue',
                'red', xs:float(3.14), 42, xs:date('1995-04-21'))"/>
    </xsl:variable>

    <xsl:text>&#xA;Here's another test of the remove() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  Our sequence ($longSequence) </xsl:text>
    <xsl:text>is:&#xA;      </xsl:text>
    <xsl:value-of select="$longSequence" separator="&#xA;      "/>
    <xsl:text>&#xA;&#xA;    Test 1. remove($longSequence, 1) </xsl:text>
    <xsl:text>= &#xA;      </xsl:text>
    <xsl:value-of select="remove($longSequence, 1)"
      separator="&#xA;      "/>
    <xsl:text>&#xA;&#xA;    Test 2. remove($longSequence, 1) =</xsl:text>
    <xsl:text>&#xA;        [We're running this test again to show the </xsl:text>
    <xsl:text>variable didn't change]&#xA;      </xsl:text>
    <xsl:value-of select="remove($longSequence, 1)"
      separator="&#xA;      "/>
    <xsl:text>&#xA;&#xA;    Test 3. remove(remove($longSequence, 7), 1) </xsl:text>
    <xsl:text>= &#xA;      </xsl:text>
    <xsl:value-of select="remove(remove($longSequence, 7), 1)"
      separator="&#xA;      "/>

    <xsl:text>&#xA;&#xA;    Creating a new variable based on </xsl:text>
    <xsl:text>remove($longSequence, 1): &#xA;      </xsl:text>
    <xsl:text>updatedSequence = remove($longSequence, 1): &#xA;      </xsl:text>
    <xsl:variable name="updatedSequence" as="item()*"
      select="remove($longSequence, 1)"/>
    <xsl:text>&#xA;    Test 4. $updatedSequence = &#xA;      </xsl:text>
    <xsl:value-of select="$updatedSequence" separator="&#xA;      "/>

  </xsl:template>

</xsl:stylesheet>
```

Результат (довольно длинный) выглядит так:

```
Here's another test of the remove() function:

  Our sequence ($longSequence) is:
      3
      4
      5
      2006-07-25-04:00
      03:14:18.783-04:00
      8
      blue
      red
      3.14
      42
      1995-04-21

    Test 1. remove($longSequence, 1) =
      4
      5
      2006-07-25-04:00
      03:14:18.783-04:00
      8
      blue
```

```
    red
    3.14
    42
    1995-04-21

Test 2. remove($longSequence, 1) =
    [We're running this test again to show the variable didn't change]
    4
    5
    2006-07-25-04:00
    03:14:18.783-04:00
    8
    blue
    red
    3.14
    42
    1995-04-21

Test 3. remove(remove($longSequence, 7), 1) =
    4
    5
    2006-07-25-04:00
    03:31:45.859-04:00
    8
    red
    3.14
    42
    1995-04-21

Creating a new variable based on remove($longSequence, 1):
    updatedSequence = remove($longSequence, 1):

Test 4. $updatedSequence =
    4
    5
    2006-07-25-04:00
    03:14:18.783-04:00
    8
    blue
    red
    3.14
    42
    1995-04-21
```

Таблица стилей сначала выводит список объектов в последовательности. В первом тесте удаление первого объекта возвращает последовательность, из которой был удален первый объект (3). Затем `remove()` снова вызывается для той же последовательности; мы получаем тот же результат. Помните, что переменные в XSLT не изменяются; они могут инициализироваться, но не изменяться. Мы можем вызывать `remove($longSequence, 1)` столько раз, сколько захотим, но результат всегда будет одним и тем же. Чтобы «удалить» объект навсегда, необходимо сохранить последовательность, возвращаемую `remove()`, в другой переменной.

В третьем тесте вложенные вызовы `remove()` используются для удаления первого и седьмого объектов. Если бы потребовалось удалить первые два объекта, было бы эффективнее использовать конструкцию `subsequence($longSequence,3)`. Но так как в нашем примере удаляемые объекты не являются смежными, приходится применять вложенные вызовы `remove()`.

В четвертом примере результаты вызова remove() сохраняются в переменной. Переменная также относится к типу item()*. Существуют три способа создания новых переменных:

```
<!-- Простейшее решение -->
<xsl:variable name="updatedSequence1" as="item()*"
  select="remove($longSequence, 1)"/>

<!-- Работает... -->
<xsl:variable name="updatedSequence2" as="item()*">
  <xsl:sequence select="remove($longSequence, 1)"/>
</xsl:variable>

<!-- Не работает! -->
<xsl:variable name="updatedSequence3" as="item()*">
  <xsl:value-of select="remove($longSequence, 1)"/>
</xsl:variable>
```

Первый способ проще всего; именно он используется в нашей таблице стилей. Второй способ тоже работает, он может пригодиться при включении нескольких объектов в последовательность. *Третий способ не работает*. Элемент <xsl:value-of> вставляет *строковое значение* подпоследовательности с разделением значений пробелами. В результате создается последовательность из одного объекта:

```
$updatedSequence3 =
4 5 2006-07-25-04:00 03:20:35.815-04:00 8 blue red 3.14 42 1995-04-21
```

Решение с <xsl:value-of> выглядит разумно, однако оно не работает – используйте <xsl:sequence>. (Я упоминаю об этом здесь, потому что сам когда-то совершил такую же ошибку; надеюсь, вы вынесете урок из моей рассеянности.)

Последний пример демонстрирует вызов remove() с пустой последовательностью:

```
<?xml version="1.0"?>
<!-- remove3.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <xsl:variable name="emptySequence" as="item()*">
      <xsl:sequence select="()"/>
    </xsl:variable>

    <xsl:text>&#xA;Here's a final test of the remove() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  Our sequence ($emptySequence) is </xsl:text>
    <xsl:text>empty:&#xA;       empty($emptySequence) = </xsl:text>
    <xsl:value-of select="empty($emptySequence)"/>
    <xsl:text>&#xA;&#xA;       remove($emptySequence, 1) = </xsl:text>
    <xsl:value-of select="remove($emptySequence, 1)"/>
    <xsl:text>&#xA;&#xA;       empty(remove($emptySequence, 1)) </xsl:text>
    <xsl:text>= </xsl:text>
    <xsl:value-of select="empty(remove($emptySequence, 1))"/>

  </xsl:template>

</xsl:stylesheet>
```

Последний набор результатов:

```
Here's a final test of the remove() function:

  Our sequence ($emptySequence) is empty:
      empty($emptySequence) = true

      remove($emptySequence, 1) =

      empty(remove($emptySequence, 1)) = true
```

[2.0] replace()

Получает исходную строку, регулярное выражение и строку замены; заменяет все совпадения регулярного выражения в исходной строке экземплярами строки замены.

Синтаксис

```
xs:string replace($input as xs:string?, $pattern as xs:string,
                  $replacement as xs:string)
xs:string replace($input as xs:string?, $pattern as xs:string,
                  $replacement as xs:string, $flags as xs:string)
```

Ввод

Три строки – исходная, регулярное выражение и строка замены. Имеется также необязательная четвертая строка с флагами, управляющими обработкой регулярного выражения.

Совпадение регулярного выражения со строкой нулевой длины приводит к фатальной ошибке. За дополнительной информацией обращайтесь к приложению Е.

Вывод

Обновленная строка, в которой все совпадения регулярного выражения были заменены экземплярами строки замены.

Более подробное описание того, как работает `replace()`:

- Если регулярное выражение не имеет совпадений в исходной строке, то исходная строка возвращается в неизменном виде.
- Если регулярное выражение имеет два перекрывающихся совпадения в исходной строке, заменяется только первое совпадение.
- При поиске совпадений для регулярных выражений не используются схемы упорядочения; сравниваются коды символов в Юникоде. Ситуации, в которых разные символы считаются одинаковыми в национальных языках, не учитываются.
- И исходная строка, и строка замены могут иметь нулевую длину. Если исходная строка имеет нулевую длину, возвращается строка нулевой длины. Если строка замены имеет нулевую длину, все совпадения регулярного выражения удаляются. Регулярное выражение не может быть строкой нулевой длины и не может совпадать со строкой нулевой длины (другими словами, `matches("", $pattern, $replacement)` не может быть равно `true`). Если длина исходной строки отлична от нуля, сохраненная подстрока *может* иметь нулевую длину.

Если выражение `matches("", $pattern, $replacement)` истинно, происходит ошибка. Например, при использовании шаблона `'.?'` с любой входной строкой происходит ошибка, потому что шаблон совпадает со строкой нулевой длины.

- В отличие от регулярных выражений, используемых в элементе `<xsl:analyze-string>`, фигурные скобки (`{` и `}`) не удваиваются. (Фигурные скобки, используемые в атрибуте `regex` элемента `<xsl:analyze-string>`, должны удваиваться, чтобы они не интерпретировались как шаблоны значений атрибутов.)
- К строке замены предъявляется ряд особых требований:
  - Литеральный знак доллара (`$`) должен экранироваться в виде `\$`, а литеральная обратная косая черта (`\`) – в виде `\\`. Если перед знаком доллара не стоит обратная косая черта или за ним не следует хотя бы одна цифра (`[0-9]`), происходит ошибка. Ошибка происходит и в том случае, если обратная косая черта не является частью пары обратных косых черт (`\\`) или экранированного знака доллара (`\$`).
  - В строке замены можно использовать запись `$N` для обозначения части исходной строки, совпадающей с N-й группой регулярного выражения в круглых скобках (по аналогии с функцией `regex-group()`). В нашем примере будет использоваться следующее регулярное выражение для идентификации номеров кредитных карт из 16 цифр:

    ```
    ([0-9]{4})-([0-9]{4})-([0-9]{4})-([0-9]{4})
    ```

    Выражение содержит четыре группы в круглых скобках. (Дефисы между скобками не принадлежат ни к одной группе.) Для номера кредитной карты `1234-5678-9101-1121` обозначение `$1` соответствует `1234`, обозначение `$2` – `5678`, `$3` – `9101`, а `$4` – `1121`.
  - Запись `$0` обозначает всю строку, совпавшую с регулярным выражением. В нашем примере это вся исходная строка, но так бывает не всегда.
  - Если группа регулярного выражения не имеет совпадения в исходной строке, ее значение является строкой нулевой длины.
  - Если вы используете выражение `$N` с одной цифрой, и эта цифра больше количества групп в регулярном выражении, возвращается строка нулевой длины. В нашем примере запись `$7` будет соответствовать строке нулевой длины.
  - При использовании записи `$N` процессор считает, что каждая последующая цифра после знака `$` относится к группе в регулярном выражении. Если это не так, он использует максимально возможное количество цифр, а остальные цифры записываются в выходной документ.

    Например, для записи `$417` в нашем примере будет сгенерирована строка `112117`. Здесь `$4` – самое длинное выражение, для которого имеется совпадение, поэтому остальные цифры (`17`) выводятся вместе с совпадением. Если бы регулярное выражение содержало 41 группу, запись `$417` интерпретировалась бы как `$41`, за которой следует цифра `7`.
- Если регулярное выражение определяет более одной альтернативы и в некоторой позиции ввода совпадают сразу несколько из возможных альтернатив, используется первая альтернатива в регулярном выражении. Пример из спецификации:

  ```
  replace("abcd", "(ab)|(a)", "[1=$1][2=$2]") = "[1=ab][2=]cd"
  ```

  Первая альтернатива в регулярном выражении (`ab`) считается совпадением, поэтому запись `$1` соответствует `ab`, а `$2` соответствует строке нулевой длины. Все происходит именно так, несмотря на то, что вторая альтернатива короче первой.

- Наконец, параметр `$flags` управляет обработкой регулярных выражений. Поддерживаются четыре флага:
  - `s` Регулярное выражение обрабатывается в режиме «совпадения точки со всеми символами». При наличии этого флага метасимвол «точка» (`.`) совпадает с любым символом. При стандартной обработке (без флага `s`) оператор «точка» совпадает с любым символом, *кроме* символа новой строки (`
`). Этот флаг особенно полезен при обработке строк, содержащих внутренние символы новой строки.
  - `m` Регулярное выражение обрабатывается в многострочном режиме. В режиме по умолчанию метасимвол `^` совпадает с началом всей строки, а метасимвол `$` совпадает с концом каждой строки. В многострочном режиме `^` совпадает с началом, а `$` – с концом любой логической строки внутри «физической» строки.
  - `i` Регулярное выражение обрабатывается без учета регистра символов. Регулярное выражение `"a"` совпадает как с `"a"`, так и с `"A"`.

    Учтите, что Юникод может создать массу проблем с этим флагом. Например, в спецификации XQuery 1.0 and XPath 2.0 Functions and Operators приводится пример с обозначением в Юникоде градусов по Кельвину (`K`), соответствующим букве `"K"`. Комбинация `regex="k"` и `flags="i"` совпадает как со знаком градусов по Кельвину, так и с буквами `"k"` (`k`) и `"K"` (`K`).

    Другие символы Юникода не преобразуются в буквы. Например, символ римской цифры `I` (`ࡰ`) внешне похож на букву `I`, но не преобразуется в нее.
  - `x` Перед поиском совпадения из регулярного выражения удаляются все символы пропусков (`	`, `
`, `
` и ` `). Другими словами, с флагом `x` регулярные выражения `"John Smith"` и `"JohnSmith"` эквивалентны. Этот флаг особенно удобен при разбиении длинного регулярного выражения на несколько строк, чтобы его было легче читать.

  Флаги могут комбинироваться в любом порядке. Наборы параметров `'xis'` и `'six'` работают абсолютно одинаково.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.6 «String Functions that Use Pattern Matching».

Пример

Таблица стилей с несколькими вариантами использования функции `replace()`:

```
<?xml version="1.0"?>
<!-- replace.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <xsl:variable name="string1"
      select="concat('Now is the time for all good men and ',
              'women to aid the party.')"/>
```

```
    <xsl:variable name="string2" as="xs:string"
      select="'Visa # 1234-5678-9101-1121'"/>

    <xsl:text>&#xA;Here's a test of the replace() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  $string1 = &#xA;    </xsl:text>
    <xsl:value-of select="$string1"/>

    <xsl:text>&#xA;&#xA;  Test 1. replace($string1, </xsl:text>
    <xsl:text>'men', 'boys') = &#xA;    </xsl:text>
    <xsl:value-of select="replace($string1, 'men', 'boys')"/>

    <xsl:text>&#xA;&#xA;  Test 2. replace($string1, </xsl:text>
    <xsl:text>' men', ' boys') = &#xA;    </xsl:text>
    <xsl:value-of select="replace($string1, ' men', 'boys')"/>

    <xsl:text>&#xA;&#xA;  Test 3. replace($string1, </xsl:text>
    <xsl:text>'wombats', 'weasels') = &#xA;    </xsl:text>
    <xsl:value-of select="replace($string1, 'wombats', 'weasels')"/>

    <xsl:text>&#xA;&#xA;  $string2 = &#xA;    </xsl:text>
    <xsl:value-of select="$string2"/>

    <xsl:text>&#xA;&#xA;  Test 4. replace($string2, </xsl:text>
    <xsl:text>&#xA;                  </xsl:text>
    <xsl:text>'([0-9]{4}-)([0-9]{4}-)([0-9]{4}-)([0-9]{4})', </xsl:text>
    <xsl:text>&#xA;                  </xsl:text>
    <xsl:text>'XXXX-XXXX-XXXX-$4') = &#xA;    </xsl:text>
    <xsl:value-of
      select="replace($string2,
              '([0-9]{4})-([0-9]{4})-([0-9]{4})-([0-9]{4})',
              'XXXX-XXXX-XXXX-$4')"/>

    <xsl:text>&#xA;&#xA;  Test 5. replace($string2, </xsl:text>
    <xsl:text>&#xA;                  </xsl:text>
    <xsl:text>'([0-9]{4}-)([0-9]{4}-)([0-9]{4}-)([0-9]{4})', </xsl:text>
    <xsl:text>&#xA;                  </xsl:text>
    <xsl:text>'$0 -> XXXX-XXXX-XXXX-$4') = &#xA;    </xsl:text>
    <xsl:value-of
      select="replace($string2,
              '([0-9]{4})-([0-9]{4})-([0-9]{4})-([0-9]{4})',
              '$0 -> XXXX-XXXX-XXXX-$4')"/>

  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here's a test of the replace() function:

  $string1 =
    Now is the time for all good men and women to aid the party.

  Test 1. replace($string1, 'men', 'boys') =
    Now is the time for all good boys and woboys to aid the party.

  Test 2. replace($string1, ' men', ' boys') =
    Now is the time for all good boys and women to aid the party.

  Test 3. replace($string1, 'wombats', 'weasels') =
```

```
  Now is the time for all good men and women to aid the party.

$string2 =
  Visa # 1234-5678-9101-1121

Test 4. replace($string2,
                '([0-9]{4})-([0-9]{4})-([0-9]{4})-([0-9]{4})',
                'XXXX-XXXX-XXXX-$4') =
  Visa # XXXX-XXXX-XXXX-1121

Test 5. replace($string2,
                '([0-9]{4})-([0-9]{4})-([0-9]{4})-([0-9]{4})',
                '$0 -> XXXX-XXXX-XXXX-$4') =
  Visa # 1234-5678-9101-1121 -> XXXX-XXXX-XXXX-1121
```

Тесты 1, 2 и 3 достаточно тривиальны. Тест 1 заменяет текст в исходной строке, но работает не совсем так, как нам хотелось бы. Тест 2 использует слегка измененный шаблон и строку замены для получения более осмысленных результатов. Тест 3 возвращает исходную строку без изменений, потому что шаблон не совпадает.

Тесты 4 и 5 используют поддержку частичных совпадений для отображения совпавшего текста из исходной строки. В обоих случаях функция `replace()` скрывает все цифры номера кредитной карты, кроме четырех последних. В обоих тестах используется четвертое частичное совпадение (`$4`), а в тесте 5 используется также конструкция `$0` для вывода всей совпавшей строки.

См. также

В приложении E приведено подробное описание работы регулярных выражений в XPath 2.0. Обращайтесь также к определениям следующих элементов и функций: ***[2.0]*** `<xsl:analyze-string>`, ***[2.0]*** `matches()`, ***[2.0]*** `<xsl:matching-substring>`, ***[2.0]*** `<xsl:non-matching-substring>`, ***[2.0]*** `regex-group()` и ***[2.0]*** `tokenize()`.

[2.0] resolve-QName()

Функция для конструирования полного имени (`QName`). Для имени (с возможным префиксом) в формате `xs:string` и элемента функция возвращает `QName`. Префикс имени, если он имеется, разрешается с использованием пространств имен в области видимости заданного элемента.

Синтаксис

```
xs:QName? resolve-QName($qname as xs:string?, $element as element())
```

Ввод

Значение `xs:string`, являющееся действительным именем XML (с возможным префиксом пространства имен), и элемент.

Вывод

Новое значение `xs:QName`.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 11.1 «Additional Constructor Functions for QNames».

Пример

Чтобы продемонстрировать создание нового значения `QName` **функцией** `resolve-QName()`**, мы вернемся к неоднократно использовавшемуся документу с сонетом Шекспира:**

```
<?xml version="1.0"?>
<!-- sonnet.xml -->
<sonnet type='Shakespearean'>
  <auth:author xmlns:auth="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </auth:author>
  <!-- Существует ли официальное название сонета?
       Иногда сонеты называются по первой строке. -->
  <title>Sonnet 130</title>
  <lines>
    <line>My mistress' eyes are nothing like the sun,</line>
    <line>Coral is far more red than her lips red.</line>
    <line>If snow be white, why then her breasts are dun,</line>
    <line>If hairs be wires, black wires grow on her head.</line>
    <line>I have seen roses damasked, red and white,</line>
    <line>But no such roses see I in her cheeks.</line>
    <line>And in some perfumes is there more delight</line>
    <line>Than in the breath that from my mistress reeks.</line>
    <line>I love to hear her speak, yet well I know</line>
    <line>That music hath a far more pleasing sound.</line>
    <line>I grant I never saw a goddess go,</line>
    <line>My mistress when she walks, treads on the ground.</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
  </lines>
</sonnet>
```

Таблица стилей:

```
<?xml version="1.0"?>
<!-- resolve-qname.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the resolve-QName() function:&#xA;</xsl:text>

    <xsl:variable name="testQName1" as="xs:QName"
      select="resolve-QName('auth:last-name',
              /sonnet/*:author/last-name)"/>

    <xsl:text>&#xA;  testQName1 = resolve-QName('auth:</xsl:text>
    <xsl:text>last-name', /sonnet/author/last-name)</xsl:text>
    <xsl:text>&#xA;  prefix-from-QName</xsl:text>
    <xsl:text>($testQName1) = "</xsl:text>
    <xsl:value-of select="prefix-from-QName($testQName1)"/>
    <xsl:text>"&#xA;  local-name-from-QName($testQName1) = "</xsl:text>
```

```
    <xsl:value-of select="local-name-from-QName($testQName1)"/>
    <xsl:text>"&#xA;  namespace-uri-from-QName($testQName1) = "</xsl:text>
    <xsl:value-of select="namespace-uri-from-QName($testQName1)"/>
    <xsl:text>"&#xA;</xsl:text>

    <xsl:variable name="testQName2" as="xs:QName"
      select="resolve-QName('something-else', /sonnet)"/>

    <xsl:text>&#xA;  testQName2 = resolve-QName('something-</xsl:text>
    <xsl:text>else', /sonnet)</xsl:text>
    <xsl:text>&#xA;  prefix-from-QName</xsl:text>
    <xsl:text>($testQName2) = "</xsl:text>
    <xsl:value-of select="prefix-from-QName($testQName2)"/>
    <xsl:text>"&#xA;  local-name-from-QName($testQName2) = "</xsl:text>
    <xsl:value-of select="local-name-from-QName($testQName2)"/>
    <xsl:text>"&#xA;  namespace-uri-from-QName($testQName2) = "</xsl:text>
    <xsl:value-of select="namespace-uri-from-QName($testQName2)"/>
    <xsl:text>"</xsl:text>

    <!-- Приводит к ошибке; префикс 'auth' не находится
         в области видимости элемента <sonnet>. -->
    <!--    <xsl:variable name="testQName3" as="xs:QName"
              select="resolve-QName('auth:new-element', /sonnet)"/> -->
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the resolve-QName() function:

  testQName1 = resolve-QName('auth:last-name', /sonnet/author/last-name)
  prefix-from-QName($testQName1) = "auth"
  local-name-from-QName($testQName1) = "last-name"
  namespace-uri-from-QName($testQName1) = "http://www.authors.com/"

  testQName2 = resolve-QName('something-else', /sonnet)
  prefix-from-QName($testQName2) = ""
  local-name-from-QName($testQName2) = "something-else"
  namespace-uri-from-QName($testQName2) = ""
```

При первом использовании `resolve-QName()` создается переменная типа `xs:QName`, использующая префикс `auth` в области видимости элемента `<last-name>`. Обратите внимание: префикс `auth` фактически определяется для родителя `<last-name>`, но он находится в области видимости. После создания `QName` мы используем функции `*-from-QName()` для отображения его компонентов.

Во втором примере создается значение `QName`, не имеющее пространства имен. Мы используем локальное имя, не имеющее префикса, поэтому функция `prefix-from-QName()` возвращает строку нулевой длины. Во втором аргументе `resolve-QName()` передается элемент `<sonnet>`. Наш документ не имеет пространства имен по умолчанию, поэтому `QName` тоже не входит в пространство имен.

Наконец, закомментированный код в таблице стилей пытается использовать пространство имен `auth` в корневом элементе (`/sonnet`). Пространство имен `auth` не входит в область видимости корневого элемента, поэтому вызов `resolve-QName()` приводит к ошибке.

[2.0] resolve-uri()

Преобразует относительный URI-адрес в абсолютный URI-адрес.

Синтаксис

```
xs:anyURI? resolve-uri($relative as xs:string?)
xs:anyURI? resolve-uri($relative as xs:string?, $base as xs:string)
```

Ввод

Значение `xs:string`, представляющее относительный URI-адрес. Также может передаваться необязательное значение `xs:string` с абсолютным URI-адресом. Если абсолютный URI-адрес не задан, используется базовый URI-адрес из статического контекста.

Вывод

Значение `xs:anyURI`, содержащее преобразованный URI-адрес. Если второй аргумент не указан, а в текущем контексте не определен базовый URI-адрес, происходит ошибка. Если любая из входных строк не является действительным URI-адресом, происходит ошибка. Если относительный URI-адрес является абсолютным, он возвращается без изменений. Наконец, если относительный URI-адрес является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 8 «Functions on anyURI».

Пример

Следующая таблица стилей преобразует два URI-адреса. В одном преобразовании используется базовый URI-адрес из статического контекста, а в другом передается абсолютный URI-адрес.

```
<?xml version="1.0"?>
<!-- resolve-uri.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the resolve-uri() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  The static base URI for the </xsl:text>
    <xsl:text>document root is:&#xA;    </xsl:text>
    <xsl:value-of select="static-base-uri()"/>

    <xsl:text>&#xA;&#xA;  The base URI for the </xsl:text>
    <xsl:text>document root is:&#xA;    </xsl:text>
    <xsl:value-of select="base-uri()"/>

    <xsl:text>&#xA;&#xA;  resolve-uri('doug.html') = </xsl:text>
    <xsl:text>&#xA;    </xsl:text>
    <xsl:value-of select="resolve-uri('doug.html')"/>

    <xsl:text>&#xA;&#xA;  resolve-uri('doug.html', </xsl:text>
    <xsl:text>'http://www.oreilly.com/') = </xsl:text>
```

```
    <xsl:text>&#xA;    </xsl:text>
    <xsl:value-of
      select="resolve-uri('doug.html', 'http://www.oreilly.com/')"/>

  </xsl:template>

</xsl:stylesheet>
```

(Функция `static-base-uri()` используется для вывода статического базового URI-адреса для преобразования, а функция `base-uri()` выводит свойство базового URI-адреса перед использованием функции `resolve-uri()`.) Результат выглядит так:

```
Tests of the resolve-uri() function:

  The static base URI for the document root is:
    file:/C://resolve-uri.xsl

  The base URI for the document root is:
    file:/C://chocolate.xml

  resolve-uri('doug.html') =
    file:/C:/doug.html

  resolve-uri('doug.html', 'http://www.oreilly.com/') =
    http://www.oreilly.com/doug.html
```

Эти результаты были получены при применении таблицы стилей к файлу *chocolate.xml*. Местонахождение файла в файловой системе становится базовым URI-адресом для документа. Первый тест создает URI-адрес для документа, находящегося в одном каталоге с исходным документом. Второй тест получает имя ресурса и URL-адрес сайта и объединяет их, создавая URI-адрес.

[2.0] reverse()

Возвращает последовательность, полученную перестановкой объектов исходной последовательности в обратном порядке.

Синтаксис

```
item()* reverse(item()*)
```

Ввод

Последовательность. Если исходная последовательность является пустой, то функция возваращает пустую последовательность.

Вывод

Последовательность с объектами, переставленными в обратном порядке.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.1 «General Functions and Operators on Sequences».

Пример

Следующая короткая таблица стилей создает последовательность и выводит ее, после чего выводит последовательность, полученную в результате применения функции `reverse()`:

```
<?xml version="1.0"?>
<!-- reverse.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:datatest="http://www.oreilly.com">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <xsl:variable name="testSequence" as="item()*">
      <xsl:sequence
        select="(3, 4, 5, current-date(), current-time(), 8, 'blue',
                 'red', xs:float(3.14), 42, xs:date('1995-04-21'))"/>
    </xsl:variable>

    <xsl:text>&#xA;Here is a test of the reverse() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  Our original sequence is:&#xA;&#xA;    </xsl:text>
    <xsl:value-of select="$testSequence" separator="&#xA;    "/>

    <xsl:text>&#xA;&#xA;  Passing our sequence to reverse() </xsl:text>
    <xsl:text>gives us:&#xA;&#xA;    </xsl:text>

    <xsl:value-of select="reverse($testSequence)" separator="&#xA;    "/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here is a test of the reverse() function:

  Our original sequence is:

    3
    4
    5
    2006-07-22-04:00
    13:48:23.256-04:00
    8
    blue
    red
    3.14
    42
    1995-04-21

  Passing our sequence to reverse() gives us:

    1995-04-21
    42
    3.14
    red
    blue
    8
    13:48:23.256-04:00
    2006-07-22-04:00
    5
    4
    3
```

Не забывайте, что переменные не изменяются, поэтому исходная последовательность продолжает существовать. Если вы хотите использовать обращенную последовательность в будущем, сохраните ее в новой переменной.

[2.0] root()

Возвращает корень дерева, к которому принадлежит узел.

Синтаксис

```
node() root()
node() root(node()?)
```

Ввод

Необязательный узел. Если узел не задан, используется контекстный узел.

Вывод

Корень дерева, к которому принадлежит узел (или контекстный узел для вызова `root()`). Чаще всего это узел документа, но есть и исключения.

Если при вызове передается аргумент, содержащий пустую последовательность, функция возвращает пустую последовательность. Кроме того, если аргумент сам является узлом документа, функция просто возвращает его.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 14 «Functions and Operators on Nodes».

Пример

Для тестирования функции `root()` мы снова воспользуемся списком любимых музыкальных альбомов:

```
<?xml version="1.0"?>
<!-- favorites.xml -->
<list>
  <title>A few of my favorite albums</title>
  <listitem>A Love Supreme</listitem>
  <listitem>Beat Crazy</listitem>
  <listitem>Here Come the Warm Jets</listitem>
  <listitem>Kind of Blue</listitem>
  <listitem>London Calling</listitem>
  <listitem>Remain in Light</listitem>
  <listitem>The Joshua Tree</listitem>
  <listitem>The Indestructible Beat of Soweto</listitem>
</list>
```

Таблица стилей для тестирования функции `root()`:

```
<?xml version="1.0"?>
<!-- root.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:ora="http://www.oreilly.com">
```

```
<xsl:output method="text"/>

<xsl:template match="/">
  <xsl:text>&#xA;Tests of the root() function:</xsl:text>

  <xsl:variable name="testTree" as="node()*">
    <report month="8" year="2006">
      <title>Chocolate bar sales</title>
      <brand>
        <name>Lindt</name>
        <units>27408</units>
      </brand>
    </report>
    <otherReport month="9" year="2006">
      <title>Chocolate bar sales</title>
      <brand>
        <name>Callebaut</name>
        <units>8203</units>
      </brand>
    </otherReport>
  </xsl:variable>

  <xsl:text>&#xA;&#xA;  1. root(//listitem[3]):</xsl:text>
  <xsl:text>&#xA;</xsl:text>
  <xsl:value-of
    select="ora:doc-node-test(root(//listitem[3]))"/>

  <xsl:text>&#xA;&#xA;  2. root(subsequence($testTree, 1, 1)):</xsl:text>
  <xsl:text>&#xA;</xsl:text>
  <xsl:value-of
    select="ora:doc-node-test(root(subsequence($testTree, 1, 1)))"/>

  <xsl:text>&#xA;&#xA;  3. root(subsequence($testTree, 2)):</xsl:text>
  <xsl:text>&#xA;</xsl:text>
  <xsl:value-of
    select="ora:doc-node-test(root(subsequence($testTree, 2, 1)))"/>

  <xsl:text>&#xA;&#xA;  4. root():</xsl:text>
  <xsl:text>&#xA;</xsl:text>
  <xsl:value-of
    select="ora:doc-node-test(root())"/>

</xsl:template>

<xsl:function name="ora:doc-node-test" as="xs:string">
  <xsl:param name="node" as="node()*"/>
  <xsl:choose>
    <xsl:when test="empty($node)">
      <xsl:value-of select="'       [empty sequence]'"/>
    </xsl:when>

    <xsl:when test="$node instance of document-node()">
      <xsl:value-of
        select="concat('       This IS a document node.&#xA;',
                '       root node''s name: &lt;',
                node-name($node/*[1]),
                '&gt;')"/>
    </xsl:when>
    <xsl:otherwise>
```

```
        <xsl:value-of
          select="concat('       This IS NOT a document node.&#xA;',
                  '       root node''s name: &lt;',
                  node-name($node),
                  '&gt;')"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:function>

</xsl:stylesheet>
```

Мы определяем переменную `testTree`, содержащую два элемента; документ XML не может иметь два корневых элемента, но для переменной это разрешено. Переменная используется для демонстрации вызовов `root()`, не возвращающих узел документа.

Таблица стилей использует элемент `<xsl:function>` для выделения кода обработки различных корневых элементов. Для заданной последовательности функция `ora:doc-node-test()` сначала проверяет, является ли последовательность пустой. (Например, если использовать таблицу стилей для преобразования документа XML, не содержащего трех элементов `<listitem>`, последовательность будет пустой.)

Если последовательность не пуста, функция `document-node()` проверяет, является ли узел узлом документа. Если проверка дает положительный результат, выводится имя его первого дочернего элемента. Формально узел документа может содержать более одного дочернего узла, поэтому для выбора первого дочернего узла используется выражение `$node/*[1]`. (Пример узла документа с несколькими дочерними элементами приведен в описании элемента *[2.0]* `<xsl:document>` в приложении А.) Если узел не является узлом документа, он просто передается функции `node-name()`.

Чтобы сделать таблицу стилей более надежной, выражение XPath можно было бы записать в следующем виде:

```
<xsl:when test="($node treat as node()) instance of document-node()">
```

Переменная `$node` относится к типу `node()*`; инструкция `treat as` преобразует ее к типу `node()`. Можно было бы также воспользоваться функцией `exactly-one()` и убедиться в том, что последовательность содержит только один объект. Таблица стилей гарантирует, что функция `ora:doc-node-test()` будет вызываться только для синглетов.

Разобравшись с функцией, рассмотрим четыре теста функции `root()`. Первый тест `root()` получает корневой элемент третьего элемента `<listitem>` в контекстном объекте. Во втором и третьем вызовах `root()` используются два корневых узла переменной `$testTree`. Хотя оба узла являются корневыми, ни один из них не является узлом документа. При последнем вызове `root()` возвращает корень контекстного объекта, т. е. корневой узел преобразуемого документа XML.

Таблица стилей выдает следующий результат:

```
Tests of the root() function:

  1. root(//listitem[3]):
       This IS a document node.
       root node's name: <list>

  2. root(subsequence($testTree, 1, 1)):
       This IS NOT a document node.
       root node's name: <report>
```

```
3. root(subsequence($testTree, 2)):
     This IS NOT a document node.
     root node's name: <otherReport>

4. root():
     This IS a document node.
     root node's name: <list>
```

round()

Возвращает целое число, ближайшее к аргументу.

Синтаксис

```
[1.0] number? round(number?)
[2.0] numeric? round(numeric?)
```

Ввод

[1.0] Число. Если аргумент не является числом, он преобразуется в число так, как при обработке функцией `number()`.

[2.0] В XSLT 2.0 аргумент `round()` должен относиться к числовому типу (`xs:integer`, `xs:float`, `xs:decimal`, `xs:double` или производному от них). Если аргумент не является числом, процессор XSLT выдает ошибку. Например, если аргумент является нетипизованным значением, прочитанным из документа XML, не использующего схему, его необходимо преобразовать в числовое значение перед передачей `round()`.

Вывод

Целое, ближайшее к аргументу. Если два числа находятся на равном расстоянии от аргумента (`1` и `2` равноудалены от `1.5`), возвращается число, ближайшее к положительной бесконечности. Различные значения аргумента обрабатываются по следующим правилам:

- Для положительной бесконечности возвращается положительная бесконечность.
- Для отрицательной бесконечности возвращается отрицательная бесконечность.
- Для положительного нуля возвращается положительный нуль.
- Для отрицательного нуля возвращается отрицательный нуль.
- Если аргумент лежит в диапазоне от `0` до `-0.5`, возвращается отрицательный нуль.
- Если аргумент равен `NaN` («нечисло»), функция `round()` возвращает `NaN`.

[2.0] В XSLT 2.0 выходное значение относится к тому же типу данных, что и входное значение. Другими словами, функция `round(xs:integer)` возвращает `xs:integer`, а `round(xs:float)` возвращает `xs:float`.

Где определяется

[1.0] XPath, раздел 4.4 «Number Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 6.4 «Functions on Numeric Values».

Пример

Следующая таблица стилей демонстрирует результаты вызова функции `round()` для разных значений. Входные данные берутся из следующего документа XML:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Таблица стилей с использованием функции `round()`:

```
<?xml version="1.0"?>
<!-- round1.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the round() function:&#xA;&#xA;</xsl:text>

    <xsl:text>  round(7.983) = </xsl:text>
    <xsl:value-of select="round(7.983)"/>

    <xsl:text>&#xA;  round(-7.893) = </xsl:text>
    <xsl:value-of select="round(-7.893)"/>

    <xsl:text>&#xA;  round(avg(/report/brand/units)) = </xsl:text>
    <xsl:value-of select="round(avg(/report/brand/units))"/>

    <xsl:text>&#xA;  round('blue') = </xsl:text>
    <xsl:value-of select="round('blue')"/>

  </xsl:template>

</xsl:stylesheet>
```

Обработка документа XML по этой таблице стилей дает следующий результат:

```
Tests of the round() function:

  round(7.983) = 8
  round(-7.893) = -8
  round(avg(/report/brand/units)) = 18263
  round('blue') = NaN
```

Если изменить элемент <xsl:stylesheet>, включив в него атрибут version="2.0", таблица стилей работать не будет, потому что вызов round('blue') является статической ошибкой. Можно добавить атрибут version="1.0" в элемент <xsl:value-of>, чтобы элемент обрабатывался в режиме XSLT 1.0. Можно также использовать функцию number(), чтобы преобразовать blue в число. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- round2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    ...
    <xsl:text>&#xA;  [1.0] round('blue') = </xsl:text>
            <xsl:value-of version="1.0" select="round('blue')"/>

    <xsl:text>&#xA;  round(number('blue')) = </xsl:text>
    <xsl:value-of select="round(number('blue'))"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат получается похожим:

```
Tests of the round() function:
  ...
  [1.0] round('blue') = NaN
  round(number('blue')) = NaN
```

[2.0] round-half-to-even()

Возвращает целое число, ближайшее к аргументу, за исключением любого значения, оканчивающегося на .5, которое округляется до ближайшего *четного* числа.

Синтаксис

```
numeric? round-half-to-even(numeric?)
numeric? round-half-to-even(numeric?, $precision as xs:integer)
```

Ввод

Числовое значение и необязательное значение xs:integer, определяющее точность (количество цифр) при вычислениях. Числовое значение должно относиться к типу xs:float, xs:double, xs:decimal или xs:integer. При нарушении этого условия процессор XSLT выдает ошибку.

Вывод

Целое, ближайшее к аргументу; значения, оканчивающиеся на .5, округляются до ближайшего четного числа. Если не считать этого исключения, функция round-half-to-even() работает по тем же правилам, что и round():

- Для положительной бесконечности возвращается положительная бесконечность.
- Для отрицательной бесконечности возвращается отрицательная бесконечность.
- Для положительного нуля возвращается положительный нуль.
- Для отрицательного нуля возвращается отрицательный нуль.

- Если аргумент лежит в диапазоне от 0 до -0.5, возвращается отрицательный нуль.
- Если аргумент равен NaN («нечисло»), функция возвращает NaN.

При использовании аргумента $precision процессор XSLT возвращает число, ближайшее к значению, кратному 10 в степени «*минус* $precision». Со значением -2 число округляется до ближайшего значения, кратного 100, а со значеним 2 – до ближайшего значения, кратного .01.

Выходное значение относится к тому же типу данных, что и входное значение. Другими словами, функция round-half-to-even(xs:integer) возвращает xs:integer, а round-half-to-even(xs:float) возвращает xs:float.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 6.4 «Functions on Numeric Values».

Пример

Простая таблица стилей для демонстрации работы функции round-half-to-even():

```
<?xml version="1.0"?>
<!-- round-half-to-even.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the round-half-to-even() function:&#xA;</xsl:text>

    <xsl:text>&#xA;  round-half-to-even(-0.5) = </xsl:text>
    <xsl:value-of select="round-half-to-even(-0.5)"/>

    <xsl:text>&#xA;  round-half-to-even(0.5) = </xsl:text>
    <xsl:value-of select="round-half-to-even(0.5)"/>

    <xsl:text>&#xA;  round-half-to-even(1.5) = </xsl:text>
    <xsl:value-of select="round-half-to-even(1.5)"/>

    <xsl:text>&#xA;  round-half-to-even(2.5) = </xsl:text>
    <xsl:value-of select="round-half-to-even(2.5)"/>

    <xsl:text>&#xA;  round-half-to-even(number('NaN')) = </xsl:text>
    <xsl:value-of select="round-half-to-even(number('NaN'))"/>

    <xsl:text>&#xA;  round-half-to-even</xsl:text>
    <xsl:text>(avg(/report/brand/units)) = </xsl:text>
    <xsl:value-of select="round-half-to-even(avg(/report/brand/units))"/>

    <xsl:text>&#xA;  round-half-to-even</xsl:text>
    <xsl:text>(avg(/report/brand/units), -2) = </xsl:text>
    <xsl:value-of
      select="round-half-to-even(avg(/report/brand/units), -2)"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Tests of the round-half-to-even() function:

  round-half-to-even(-0.5) = 0
```

```
round-half-to-even(0.5) = 0
round-half-to-even(1.5) = 2
round-half-to-even(2.5) = 2
round-half-to-even(avg(/report/brand/units)) = 18263
round-half-to-even(avg(/report/brand/units), -2) = 18300
```

Обратите внимание на последствия использования аргумента `$precision` в последней строке. Средний объем продаж округляется до ближайшего значения, кратного 100 (10 в степени «минус» `-2`).

[2.0] seconds-from-dateTime()

Для заданного значения `xs:dateTime` возвращает секунды.

Синтаксис

```
xs:decimal? seconds-from-dateTime(xs:dateTime?)
```

Ввод

Значение `xs:dateTime`.

Вывод

Значение `xs:decimal`, представляющее компонент секунд заданного значения `xs:dateTime`. Если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Таблица стилей для демонстрации работы функции `seconds-from-dateTime()`:

```
<?xml version="1.0"?>
<!-- seconds-from-datetime.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the seconds from an xs:dateTime:</xsl:text>
    <xsl:variable name="currentDateTime" as="xs:dateTime"
      select="current-dateTime()"/>
    <xsl:text>&#xA;&#xA;  The current date and time is: </xsl:text>
    <xsl:value-of select="$currentDateTime"/>

    <xsl:text>&#xA;&#xA;  The current seconds are: </xsl:text>
    <xsl:value-of select="seconds-from-dateTime($currentDateTime)"/>
    <xsl:text>, &#xA;    usually written as a whole number (</xsl:text>
    <xsl:value-of select="format-dateTime($currentDateTime, '[s01]')"/>
    <xsl:text>)</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей создает следующий результат:

```
Extracting the seconds from an xs:dateTime:

  The current date and time is: 2006-11-16T05:02:55.438-05:00

  The current seconds are: 55.438,
    usually written as a whole number (55)
```

См. также

Определения функций *[2.0]* `day-from-dateTime()`, *[2.0]* `format-dateTime()`, *[2.0]* `hours-from-dateTime()`, *[2.0]* `minutes-from-dateTime()`, *[2.0]* `month-from-dateTime()`, *[2.0]* `timezone-from-dateTime()` и *[2.0]* `year-from-dateTime()`.

[2.0] seconds-from-duration()

Возвращает количество секунд во временном промежутке `xs:duration`.

Синтаксис

```
xs:decimal? seconds-from-duration(xs:duration?)
```

Ввод

Значение `xs:duration`.

Вывод

Значение `xs:decimal`, представляющее компонент секунд в заданном промежутке `xs:duration`. Учтите, что для `xs:yearMonthDuration` эта функция всегда возвращает `0`, потому что `xs:yearMonthDuration` не содержит компонент секунд. Кроме того, если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Следующая таблица стилей демонстрирует работу функции `seconds-from-duration()` для всех трех типов промежутков:

```
<?xml version="1.0"?>
<!-- seconds-from-duration.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the seconds component from durations:</xsl:text>

    <xsl:variable name="sampleDuration" as="xs:duration"
      select="xs:duration('P3Y8M2DT4H23M12.2S')"/>
    <xsl:variable name="sampleYearMonthDuration" as="xs:yearMonthDuration"
      select="xs:yearMonthDuration('P3Y8M')"/>
```

```
    <xsl:variable name="sampleDayTimeDuration" as="xs:dayTimeDuration"
      select="xs:dayTimeDuration('P2DT4H23M12.2S')"/>

    <xsl:text>&#xA;&#xA;  A sample xs:duration: </xsl:text>
    <xsl:value-of select="$sampleDuration"/>
    <xsl:text>&#xA;    The seconds component of this duration is </xsl:text>
    <xsl:value-of select="seconds-from-duration($sampleDuration)"/>
    <xsl:text>.</xsl:text>

    <xsl:text>&#xA;&#xA;  A sample xs:yearMonthDuration: </xsl:text>
    <xsl:value-of select="$sampleYearMonthDuration"/>
    <xsl:text>&#xA;    The seconds component of this duration is </xsl:text>
    <xsl:value-of select="seconds-from-duration($sampleYearMonthDuration)"/>
    <xsl:text>.</xsl:text>

    <xsl:text>&#xA;&#xA;  A sample xs:dayTimeDuration: </xsl:text>
    <xsl:value-of select="$sampleDayTimeDuration"/>
    <xsl:text>&#xA;    The seconds component of this duration is </xsl:text>
    <xsl:value-of select="seconds-from-duration($sampleDayTimeDuration)"/>
    <xsl:text>.</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Extracting the seconds component from durations:

  A sample xs:duration: P3Y8M2DT4H23M12.2S
    The seconds component of this duration is 12.2.

  A sample xs:yearMonthDuration: P3Y8M
    The seconds component of this duration is 0.

  A sample xs:dayTimeDuration: P2DT4H23M12.2S
    The seconds component of this duration is 12.2.
```

Как упоминалось ранее, вызов `seconds-from-duration()` **для значения** `xs:yearMonthDuration` **возвращает** `0`**.**

См. также

Определения функций ***[2.0]*** `days-from-duration()`**,** ***[2.0]*** `hours-from-duration()`**,** ***[2.0]*** `minutes-from-duration()`**,** ***[2.0]*** `months-from-duration()` **и** ***[2.0]*** `years-from-duration()`**.**

[2.0] seconds-from-time()

Для заданного значения `xs:time` **возвращает секунды.**

Синтаксис

```
xs:decimal? seconds-from-time(xs:time?)
```

Ввод

Значение `xs:time`**.**

Вывод

Значение `xs:decimal`**, представляющее компонент секунд заданного значения** `xs:time`**. Если аргумент является пустой последовательностью, функция возвращает пустую последовательность.**

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Таблица стилей для демонстрации работы функции `seconds-from-time()`:

```
<?xml version="1.0"?>
<!-- seconds-from-time.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the seconds component from an xs:time:</xsl:text>
    <xsl:variable name="currentTime" as="xs:time" select="current-time()"/>
    <xsl:text>&#xA;&#xA;  The current time is: </xsl:text>
    <xsl:value-of select="$currentTime"/>

    <xsl:text>&#xA;&#xA;  The current seconds are: </xsl:text>
    <xsl:value-of select="seconds-from-time($currentTime)"/>
    <xsl:text>, &#xA;    usually written as a whole number (</xsl:text>
    <xsl:value-of select="format-time($currentTime, '[s01]')"/>
    <xsl:text>)</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Extracting the seconds component from an xs:time:

  The current time is: 05:07:40.307-05:00

  The current seconds are: 40.307,
    usually written as a whole number (40)
```

Первый результат сгенерирован функцией `seconds-from-time()`, а второй – функцией `format-time()`.

См. также

Определения функций *[2.0]* `hours-from-time()`, *[2.0]* `format-time()`, *[2.0]* `minutes-from-time()`, *[2.0]* `seconds-from-time()` и *[2.0]* `timezone-from-time()`.

starts-with()

Определяет, начинается ли первый строковый аргумент со второго аргумента.

Синтаксис

```
[1.0] boolean starts-with(string, string)
[2.0] xs:boolean starts-with(xs:string, xs:string)
[2.0] xs:boolean starts-with(xs:string, xs:string, $collation as xs:string)
```

Ввод

Две строки.

[2.0] В XSLT 2.0 поддерживается необязательный третий аргумент – имя схемы упорядочения, определяющей способ сравнения строк.

Вывод

Если оба аргумента не являются строками нулевой длины, а первая строка начинается со второй, `starts-with()` возвращает логическое значение `true`; в остальных случаях возвращается `false`. Если вторая строка имеет нулевую длину, `starts-with()` возвращает `true`. Если обе строки, первая и вторая, имеют нулевую длину, `starts-with()` возвращает `true`. Если первая строка является строкой нулевой длины, а вторая – нет, `starts-with()` возвращает `false`.

Где определяется

[1.0] XPath, раздел 4.2 «String Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.5 «Functions Based on Substring Matching».

Пример

Для демонстрации работы функции `starts-with()` будет использоваться следующий документ XML:

```
<?xml version="1.0"?>
<!-- favorites.xml -->
<list>
  <title>A few of my favorite albums</title>
  <listitem>A Love Supreme</listitem>
  <listitem>Beat Crazy</listitem>
  <listitem>Here Come the Warm Jets</listitem>
  <listitem>Kind of Blue</listitem>
  <listitem>London Calling</listitem>
  <listitem>Remain in Light</listitem>
  <listitem>The Joshua Tree</listitem>
  <listitem>The Indestructible Beat of Soweto</listitem>
</list>
```

Следующая таблица стилей выводит содержимое всех элементов `<listitem>`, начинающихся со строки «The»:

```
<?xml version="1.0"?>
<!-- starts-with.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the starts-with() </xsl:text>
    <xsl:text>function:&#xA;&#xA;</xsl:text>
    <xsl:for-each select="list/listitem[starts-with(., 'The')]">
      <xsl:number count="listitem" format="1. "/>
      <xsl:value-of select="."/>
      <xsl:text>&#xA;</xsl:text>
```

```
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
A test of the starts-with() function:

7. The Joshua Tree
8. The Indestructible Beat of Soweto
```

Функция `starts-with()` используется в предикатном выражении XPath для выбора всех узлов `<listitem>`, начинающихся с «The».

Элемент `<xsl:number>` указывает позицию каждого объекта в исходном списке. При использовании функции `position()` были бы сгенерированы метки `1` и `2`, потому что в элементе `<xsl:for-each>` обрабатываются только два узла.

[2.0] Функция `starts-with` входит в ограниченный набор функций, которым может передаваться необязательный аргумент со схемой упорядочения. Способ определения схемы упорядочения зависит от процессора XSLT. В Saxon для этого используется URI-адрес, включающий имя класса Java, выполняющего упорядочение. В следующем примере сравниваются немецкие слова *Straße* и *Strasse*. Со схемой упорядочения по умолчанию эти два слова различаются; со схемой упорядочения для немецкого языка они совпадают. Таблица стилей выглядит так:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- starts-with2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="string1" select="'Stra&#xDF;e'"/>
    <xsl:variable name="string2" select="'Strasse'"/>

    <xsl:text>&#xA;A test of the starts-with() function:&#xA;</xsl:text>

    <xsl:text>  starts-with('</xsl:text>
    <xsl:value-of select="$string1"/>
    <xsl:text>', '</xsl:text>
    <xsl:value-of select="$string2"/>
    <xsl:text>') = </xsl:text>
    <xsl:value-of select="starts-with($string1, $string2)"/>
    <xsl:text>&#xA;</xsl:text>

    <xsl:text>  starts-with('</xsl:text>
    <xsl:value-of select="$string1"/>
    <xsl:text>', '</xsl:text>
    <xsl:value-of select="$string2"/>
    <xsl:text>', [German collation]) = </xsl:text>
    <xsl:value-of
      select="starts-with($string1, $string2,
              concat('http://saxon.sf.net/collation?',
                     'class=com.oreilly.xslt.GermanCollation;'))"/>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
A test of the starts-with() function:
  starts-with('Straße', 'Strasse') = false
  starts-with('Straße', 'Strasse', [German collation]) = true
```

С определенной нами пользовательской схемой упорядочения два варианта написания Strasse считаются одинаковыми; со схемой упорядочения по умолчанию они различаются. (Чтобы листинг поместился на странице, мы объединили две половины URI-адреса схемы упорядочения Saxon при помощи функции concat().) За дополнительной информацией обращайтесь к разделу «Функция document() и сортировка» главы 8.

[2.0] static-base-uri()

Возвращает значение свойства базового URI-адреса из статического контекста. Свойство базового URI-адреса может изменяться в зависимости от элемента XML, но static-base-uri() возвращает одно и то же значение независимо от текущего местоположения во входном документе. В Saxon-J 9.0.0.3 статическим базовым URI-адресом является базовый URI-адрес таблицы стилей. Altova XML работает аналогично, хотя и использует несколько иной формат URI-адреса.

Синтаксис

```
xs:anyURI? static-base-uri()
```

Ввод

Нет.

Вывод

Значение xs:anyURI, представляющее статический базовый URI-адрес. Если свойство базового URI-адреса не определено в статическом контексте, static-base-uri() возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 16 «Context Functions».

Пример

Работа функции static-base-uri() будет продемонстрирована на примере более раннего документа, в котором атрибут xml:base использовался для определения разных базовых URI-адресов в разных точках документа:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- xmlbase.xml -->
<cars>
  <manufacturer name="Chevrolet"
    xml:base="http://www.chevrolet.com/">
    <car>Cavalier</car>
    <car>Corvette</car>
    <car>Impala</car>
    <car>Malibu</car>
  </manufacturer>
  <manufacturer name="Ford"
```

```
    xml:base="http://www.ford.com/">
    <car>Pinto</car>
    <car>Mustang</car>
    <car>Taurus</car>
  </manufacturer>
  <manufacturer name="Volkswagen"
    xml:base="http://www.vw.com/">
    <car>Beetle</car>
    <car>Jetta</car>
    <car>Passat</car>
    <car>Touraeg</car>
  </manufacturer>
</cars>
```

Далее приводится таблица стилей. Мы вызываем функцию `static-base-uri()` **для корня документа, а затем для элементов** `<manufacturer>`**, изменяющих базовый URI-адрес:**

```
<?xml version="1.0"?>
<!-- static-base-uri.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the static-base-uri() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  The static base URI for the </xsl:text>
    <xsl:text>document root is:&#xA;    </xsl:text>
    <xsl:value-of select="static-base-uri()"/>
    <xsl:text>&#xA;&#xA;</xsl:text>

    <xsl:for-each select="/cars/manufacturer">
      <xsl:text>  The static base URI for manufacturer </xsl:text>
      <xsl:value-of select="@name"/>
      <xsl:text> is: &#xA;    </xsl:text>
      <xsl:value-of select="static-base-uri()"/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the static-base-uri() function:

  The static base URI for the document root is:
    file:/C://static-base-uri.xsl
  The static base URI for manufacturer Chevrolet is:
    file:/C://static-base-uri.xsl
  The static base URI for manufacturer Ford is:
    file:/C://static-base-uri.xsl
  The static base URI for manufacturer Volkswagen is:
    file:/C://static-base-uri.xsl
```

Видно, что функция `static-base-uri()` **всегда возвращает одно и то же значение (URI-адрес таблицы стилей) независимо от того, какая часть документа XML обрабатывается в настоящий момент или определены ли атрибуты** `xml:base`**.**

string()

Возвращает строковое представление аргумента.

Синтаксис

```
[1.0] string string(object)
[2.0] xs:string string(item()?)
```

Ввод

Объект. Аргумент преобразуется в строку по правилам, описанным в следующем подразделе.

Вывод

[1.0] Строка. Аргумент преобразуется в строку по следующим правилам:

- Если аргумент является набором узлов, первый узел набора преобразуется в строку. (Первым узлом в наборе считается узел, следующий первым в порядке документа.) Для элемента или корневого узла это означает все его текстовое содержимое, а для атрибута – его значение.
- Если аргумент является числом, он преобразуется в строку по следующим правилам:
  - Значение `NaN` пребразуется в строку `"NaN"`.
  - Положительный нуль преобразуется в строку `"0"`.
  - Отрицательный нуль преобразуется в строку `"0"`.
  - Положительная бесконечность преобразуется в строку `"Infinity"`.
  - Отрицательная бесконечность преобразуется в строку `"-Infinity"`.
  - Целое число преобразуется в свое строковое представление без десятичной точки (разделителя дробной части) и без начальных нулей. Отрицательные числа снабжаются префиксом «минус» (-).
  - Все остальные числа преобразуются в строковое представление с дробной частью – по крайней мере с одной цифрой до десятичной точки и по крайней мере с одной цифрой после нее. Отрицательные числа снабжаются префиксом «минус» (-). Целая часть не содержит начальных нулей (с возможным исключением одной обязательной цифры до десятичной точки). Дробная часть содержит столько цифр, сколько необходимо для того, чтобы отличить это число от других числовых значений, определяемых в стандарте IEEE 754 (этот стандарт используется также для типов Java `float` и `double`).
- Если аргумент является логическим значением, значение `true` представляется строкой `"true"`, а значение `false` – строкой `"false"`.
- Если аргумент относится к другому типу, он преобразуется в строку способом, специфическим для данного типа. За информацией о поддерживаемых типах и способах их преобразования в строки обращайтесь к документации процессора XSLT.

[2.0] Значение `xs:string`. Поскольку XSLT 2.0 поддерживает большее количество типов данных, чем XSLT 1.0, правила преобразования к строковому виду усложняются. В XSLT 2.0 используется следующая система правил:

- Если значение относится к типу `xs:string` или любому типу данных, производному от `xs:string`, аргумент возвращается без изменений.

- Если объект является последовательностью, он должен быть либо пустой, либо синглетной последовательностью. *Вызов* `string()` *для последовательности, содержащей более одного объекта, является ошибкой*. В этом отношении XSLT 2.0 существенно отличается от XSLT 1.0, где `string()` (и многие другие функции) просто берут первый объект набора узлов и обрабатывают его. Если вы адаптируете таблицу стилей XSLT 1.0 для XSLT 2.0, фрагмент:

  ```
  <xsl:value-of select="string(items/item/price)"/>
  ```

  придется заменить следующим фрагментом:

  ```
  <xsl:value-of select="string((items/item/price)[1])"/>
  ```

  Если функция `string()` вызывается без аргумента, используется контекстный объект. Иначе говоря, вызов `string()` эквивалентен `string(.)`. Вызов `string()` для пустой последовательности возвращает строку нулевой длины.

- Если объект является числовым значением (`xs:integer`, `xs:decimal`, `xs:float`, `xs:double` или любого производного типа), он преобразуется в строку по следующим правилам:
  - Значение `NaN` («нечисло») пребразуется в строку `"NaN"`.
  - Положительная бесконечность преобразуется в строку `"INF"`.
  - Отрицательная бесконечность преобразуется в строку `"-INF"`.
  - Положительный нуль преобразуется в строку `0`.
  - Отрицательный нуль преобразуется в строку `-0`.

  В спецификации XQuery 1.0 and XPath 2.0 Functions and Operators приведено исчерпывающее описание правил преобразования различных числовых типов данных в строки. Например, значение `xs:integer` не должно содержать разделителя дробной части при преобразовании в строку. Существуют также правила, определяющие формат мантиссы и экспоненты, обработку значений `xs:decimal` без дробной части, и много других. За полной информацией обращайтесь к спецификации.

- Если объект относится к типу `xs:date`, `xs:time` или `xs:dateTime`, строковое значение создается в формате по умолчанию. Например, значение `xs:date` в большинстве случаев представляется в формате `YYYY-MM-DD-Z` (формат `-Z` представляет часовой пояс, если он определен), `xs:time` представляется в формате `HH:MM:SS-Z` (где `Z` – часовой пояс), а `xs:dateTime` – в формате `YYYY-MM-DDTHH:MM:SS-Z`.

  Функция `string()` возвращает низкоуровневое представление этих типов; для преобразования типов даты и времени в строку удобнее использовать функции `format-date()`, `format-dateTime()` и `format-time()`.

- Если объект относится к типу `xs:dayTimeDuration`, `xs:yearMonthDuration` или базовому типу `xs:duration`, строковое значение создается в формате по умолчанию. Например, значение `xs:dayTimeDuration` представляется в виде `P2DT4H23M12.2S` (2 дня, 4 часа, 23 минуты и 12,2 секунды). Для преобразования временных промежутков в строку удобнее использовать функции `days-from-duration()`, `hours-from-duration()`, `minutes-from-duration()`, `months-from-duration()`, `seconds-from-duration()` и `years-from-duration()`.

- Если объект относится к типу `xs:anyURI`, он возвращается в строковом виде. Символы строки при этом не экранируются.

- Другие типы данных XML Schema, имеющие формат по умолчанию (обычно называемый «каноническим представлением»), преобразуются в значения `xs:string`

с использованием этого формата. Процесс преобразования в строки типов данных, не имеющих канонического представления, определяется реализацией.

Где определяется

[1.0] XPath, раздел 4.2 «String Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 2 «Accessors»; правила преобразования типов данных в строки определяются в разделе 17.1.2 «Casting to xs:string and xs:untypedAtomic».

Пример

Для тестирования функции `string()` используется документ с данными о продажах шоколада:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Функция `string()` тестируется с разнообразными аргументами:

```
<?xml version="1.0"?>
<!-- string.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the string() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  string(count(/report/brand)) = </xsl:text>
    <xsl:value-of select="string(count(/report/brand))"/>

    <xsl:text>&#xA;&#xA;  string(true()) = </xsl:text>
    <xsl:value-of select="string(true())"/>
```

```
    <xsl:text>&#xA;&#xA;  string(count(/report</xsl:text>
    <xsl:text>/brand/units[. &gt; 20000])) = </xsl:text>
    <xsl:value-of
      select="string(count(/report/brand/units[. &gt; 20000]))"/>

    <xsl:text>&#xA;&#xA;  string(/report/brand/units</xsl:text>
    <xsl:text>[. &gt; 20000][1]/../name) = </xsl:text>
    <xsl:value-of
      select="string(/report/brand/units[. &gt; 20000][1]/../name)"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Tests of the string() function:

  string(count(/report/brand)) = 5

  string(true()) = true

  string(count(/report/brand/units[. > 20000])) = 2

  string(/report/brand/units[. > 20000][1]/../name) = Lindt
```

[2.0] **Процесс преобразования значения в строку обычно достаточно тривиален. Следующая таблица стилей преобразует некоторые типы данных XSLT 2.0 в строки:**

```
<?xml version="1.0"?>
<!-- string2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;More tests of the string() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  string(xs:double(2718.28E-3)) = </xsl:text>
    <xsl:value-of select="string(xs:double(2718.28E-3))"/>

    <xsl:text>&#xA;&#xA;  string(current-time()) = </xsl:text>
    <xsl:value-of select="string(current-time())"/>

    <xsl:text>&#xA;&#xA;  string(xs:yearMonthDuration</xsl:text>
    <xsl:text>('P3Y8M')) = </xsl:text>
    <xsl:value-of
      select="string(xs:yearMonthDuration('P3Y8M'))"/>

    <xsl:text>&#xA;&#xA;  string(xs:date('1995-04-21')) = </xsl:text>
    <xsl:value-of
      select="string(xs:date('1995-04-21'))"/>

    <xsl:variable name="topBrands" as="xs:string*"
      select="/report/brand/units[. &gt; 15000]/../name/string()"/>
    <xsl:text>&#xA;&#xA;  Top-selling brands: &#xA;    </xsl:text>
    <xsl:value-of select="$topBrands" separator=", "/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
More tests of the string() function:

  string(xs:double(2718.28E-3)) = 2.71828

  string(current-time()) = 06:38:08.438-04:00

  string(xs:yearMonthDuration('P3Y8M')) = P3Y8M

  string(xs:date('1995-04-21')) = 1995-04-21

  Top-selling brands:
    Lindt, Valrhona, Ghirardelli
```

Таблица стилей использует функцию `string()` и в последней фазе выражения XPath. Выражение `/report/brand/units[. > 15000]/../name/string()` находит все элементы `<units>` со значением, большим 15000, а затем преобразует текст их одноуровневых элементов `<name>` в строки, возвращая последовательность строк. При выводе этой последовательности с использованием элемента `<xsl:value-of>` перечисляются три производителя (Lindt, Valrhona и Ghirardelli).

[2.0] string-join()

Для заданной последовательности строк и строки-разделителя возвращает строку, содержащую все строки исходной последовательности, разделенные экземплярами второй строки.

Синтаксис

```
xs:string string-join(xs:string*, xs:string)
```

Ввод

Последовательность значений `xs:string` и строка, используемая в качестве разделителя.

Вывод

Значение `xs:string`, содержащее все строки исходной последовательности, разделенные вторым аргументом. Если во втором аргументе передается строка нулевой длины, строки последовательности просто объединяются посредством конкатенации, без разделителя. Если последовательность является пустой последовательностью, `string-join()` возвращает строку нулевой длины.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.4 «Functions on String Values».

Пример

Таблица стилей для тестирования функции `string-join()`:

```
<?xml version="1.0"?>
<!-- string-join.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xsl:output method="text"/>

    <xsl:template match="/">
      <xsl:text>&#xA;A test of the string-join() function:&#xA;</xsl:text>

      <xsl:variable name="testSequence" as="xs:string*">
        <xsl:sequence
          select="('Brooklyn', 'Manhattan', 'Williamsburg',
                   'George Washington', 'Tribeca')"/>
      </xsl:variable>

      <xsl:text>&#xA;  Our sequence of strings:&#xA;    </xsl:text>
      <xsl:value-of select="$testSequence" separator="&#xA;    "/>

      <xsl:text>&#xA;&#xA;  string-join($testSequeqnce, ' - ') = </xsl:text>
      <xsl:text>&#xA;    </xsl:text>
      <xsl:value-of select="string-join($testSequence, ' - ')"/>

      <xsl:text>&#xA;&#xA;  Courtesy of the specs, here's </xsl:text>
      <xsl:text>another example:&#xA;&#xA;    Ancestors of all </xsl:text>
      <xsl:text>the elements:&#xA;      using string-join&#xA;</xsl:text>

      <xsl:text>            (for $i in unordered</xsl:text>
      <xsl:text>(ancestor-or-self::*) return name($i), </xsl:text>
      <xsl:text>&#xA;              '/'):&#xA;</xsl:text>

      <xsl:for-each select="//*">
        <xsl:text>&#xA;      </xsl:text>
        <xsl:value-of
          select="string-join(for $i in ancestor-or-self::*
                  return name($i), '/')"/>
      </xsl:for-each>
    </xsl:template>

  </xsl:stylesheet>
```

Результат:

```
  A test of the string-join() function:

    Our sequence of strings:
      Brooklyn
      Manhattan
      Williamsburg
      George Washington
      Tribeca

    string-join($testSequeqnce, ' - ') =
      Brooklyn - Manhattan - Williamsburg - George Washington - Tribeca

    Courtesy of the specs, here's another example:

      Ancestors of all the elements:
        using string-join
              (for $i in unordered(ancestor-or-self::*) return name($i),
               '/'):

        report
        report/title
        report/brand
```

```
report/brand/name
report/brand/units
report/brand
report/brand/name
report/brand/units
report/brand
report/brand/name
report/brand/units
report/brand
report/brand/name
report/brand/units
report/brand
report/brand/name
report/brand/units
```

string-length()

Возвращает количество символов в строке, переданной в аргументе функции. Если аргумент не указан, контекстный узел преобразуется в строку, и функция возвращает длину этой строки.

Синтаксис

```
[1.0] number string-length(string?)
[2.0] xs:integer string-length()
[2.0] xs:integer string-length(string?)
```

Ввод

Необязательная строка.

Вывод

Количество символов, определяемых в строке (`xs:integer` в XSLT 2.0). Если аргумент не указан, используется контекстный узел. Иначе говоря, вызовы `string-length()` и `string-length(.)` эквивалентны.

[2.0] Если строка является пустой последовательностью, возвращается значение `0`.

Где определяется

[1.0] XPath, раздел 4.2 «String Functions».

[2.0] XQuery 1.0 and XPath 2.0, раздел 7.4 «Functions on String Values».

Пример

Следующий пример демонстрирует результат вызова функции `string-length()` для различных типов аргументов. Документ XML, который будет использоваться в нашем примере:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
```

```
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Документ обрабатывается по следующей таблице стилей:

```
<?xml version="1.0"?>
<!-- string-length.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the string-length() function:</xsl:text>

    <xsl:text>&#xA;&#xA;   string-length() = </xsl:text>
    <xsl:value-of select="string-length()"/>
    <xsl:text>&#xA;   string-length(/report) = </xsl:text>
    <xsl:value-of select="string-length(/report)"/>
    <xsl:text>&#xA;   string-length(/report/brand[1]) = </xsl:text>
    <xsl:value-of select="string-length(/report/brand[1])"/>
    <xsl:text>&#xA;   string-length(/report/brand[1]/name) = </xsl:text>
    <xsl:value-of select="string-length(/report/brand[1]/name)"/>

    <xsl:text>&#xA;</xsl:text>
    <xsl:for-each select="/report/brand">
      <xsl:text>&#xA;   The name of brand #</xsl:text>
      <xsl:value-of select="position()"/>
      <xsl:text>, </xsl:text>
      <xsl:value-of select="name"/>
      <xsl:text>, has </xsl:text>
      <xsl:value-of select="string-length(name)"/>
      <xsl:text> characters.</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Результат выполнения таблицы стилей:

```
Tests of the string-length() function:

   string-length() = 168
   string-length(/report) = 168
```

```
string-length(/report/brand[1]) = 23
string-length(/report/brand[1]/name) = 5

The name of brand #1, Lindt, has 5 characters.
The name of brand #2, Callebaut, has 9 characters.
The name of brand #3, Valrhona, has 8 characters.
The name of brand #4, Perugina, has 8 characters.
The name of brand #5, Ghirardelli, has 11 characters.
```

При вызове функции `string-length()` без аргументов корневой узел (в XSLT 2.0 – узел документа) преобразуется в строку, и функция возвращает длину полученной строки. (В этой точке контекстным узлом является корневой узел.) Длина строки корневого элемента (`<report>`) имеет то же значение, так как документ не содержит текста за пределами элемента `<report>`. Не забывайте, что строковое значение любого узла представляет собой результат конкатенации всего текста этого узла и всех его потомков.

[2.0] string-to-codepoints()

Преобразует строку в последовательность кодов Юникода.

Синтаксис

```
xs:integer* string-to-codepoints(xs:string?)
```

Ввод

Значение `xs:string`.

Вывод

Последовательность значений `xs:integer`, каждое из которых представляет код символа в Юникоде. Если входная строка имеет нулевую длину или является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.2 «Functions to Assemble and Disassemble Strings».

Пример

Короткая таблица стилей для преобразования строки в последовательность целых чисел:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- string-to-codepoints.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the string-to-codepoints() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  string-to-codepoints('Lily') = </xsl:text>
    <xsl:value-of
      select="string-to-codepoints('Lily')" separator=",  " />
```

```
    <xsl:text>&#xA;  string-to-codepoints('Straße') = </xsl:text>
    <xsl:value-of
      select="string-to-codepoints('Straße')" separator=", "/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Tests of the string-to-codepoints() function:

  string-to-codepoints('Lily') = 76, 105, 108, 121
  string-to-codepoints('Straße') = 83, 116, 114, 97, 223, 101
```

[2.0] subsequence()

Возвращает смежную группу объектов из последовательности («подпоследовательность»).

Синтаксис

```
item()* subsequence($sourceSeq as item()*, $startingLoc as xs:double)
item()* subsequence($sourceSeq as item()*, $startingLoc as xs:double,
                    $length as xs:double)
```

Ввод

Последовательность и начальная позиция. Необязательный третий аргумент задает количество возвращаемых объектов. Второй и третий аргументы относятся к типу `xs:double`, потому что многие числовые операции с нетипизованными данными возвращают `xs:double`.

Некоторые подробности о двух числовых аргументах:

- Если начальная позиция больше количества объектов в последовательности, возвращается пустая последовательность.
- Если исходная последовательность является пустой последовательностью, функция возвращает пустую последовательность.

Вывод

Новая последовательность, содержащая заданные объекты.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.1 «General Functions and Operators on Sequences».

Пример

Таблица стилей для демонстрации работы функции `subsequence()`:

```
<?xml version="1.0"?>
<!-- subsequence.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>
```

```
<xsl:template match="/">

  <xsl:variable name="testSequence" as="item()*">
    <xsl:sequence select="(3, 4, 5)"/>
    <xsl:element name="currentDate">
      <xsl:value-of select="current-date()"/>
    </xsl:element>
    <xsl:element name="currentTime">
      <xsl:value-of select="current-time()"/>
    </xsl:element>
    <xsl:element name="integerTest">
      <xsl:value-of select="xs:integer(8)"/>
    </xsl:element>
    <xsl:sequence select="('blue', 'red')"/>
    <xsl:element name="floatTest1">
      <xsl:value-of select="xs:float(3.14)"/>
    </xsl:element>
    <xsl:element name="floatTest2">
      <xsl:value-of select="xs:float(42)"/>
    </xsl:element>
    <xsl:element name="dateTest">
      <xsl:value-of select="xs:date('1995-04-21')"/>
    </xsl:element>
  </xsl:variable>

  <xsl:variable name="emptySequence" as="item()*">
    <xsl:sequence select="()"/>
  </xsl:variable>

  <xsl:text>&#xA;Here is a test of the subsequence() </xsl:text>
  <xsl:text>function:&#xA;</xsl:text>

  <xsl:text>&#xA;  Our original sequence is:&#xA;&#xA;    </xsl:text>
  <xsl:value-of select="$testSequence"
    separator="&#xA;    "/>

  <xsl:text>&#xA;&#xA;  subsequence($testSequence, 8) = </xsl:text>
  <xsl:text>&#xA;    </xsl:text>
  <xsl:value-of select="subsequence($testSequence, 8)"
    separator="&#xA;    "/>

  <xsl:text>&#xA;&#xA;  subsequence($testSequence, 3, 3) = </xsl:text>
  <xsl:text>&#xA;    </xsl:text>
  <xsl:value-of select="subsequence($testSequence, 3, 3)"
    separator="&#xA;    "/>

  <xsl:text>&#xA;&#xA;  subsequence($testSequence, 1, 1) = </xsl:text>
  <xsl:text>&#xA;    </xsl:text>
  <xsl:value-of select="subsequence($testSequence, 1, 1)"
    separator="&#xA;    "/>

  <xsl:text>&#xA;&#xA;  subsequence($testSequence, -1, 3) = </xsl:text>
  <xsl:text>&#xA;    </xsl:text>
  <xsl:value-of select="subsequence($testSequence, -1, 3)"
    separator="&#xA;    "/>

  <xsl:text>&#xA;&#xA;  subsequence($testSequence, 14, 2) = </xsl:text>
  <xsl:text>&#xA;    </xsl:text>
  <xsl:value-of select="subsequence($testSequence, 14, 2)"
    separator="&#xA;    "/>
```

```
    <xsl:text>&#xA;&#xA;  subsequence($testSequence, 2.3, 1.7) = </xsl:text>
    <xsl:text>&#xA;    </xsl:text>
    <xsl:value-of select="subsequence($testSequence, 2.3, 1.7)"
      separator="&#xA;    "/>

    <xsl:text>&#xA;&#xA;  subsequence($emptySequence, 1) = </xsl:text>
    <xsl:text>&#xA;    </xsl:text>
    <xsl:value-of select="subsequence($emptySequence, 1)"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here is a test of the subsequence() function:

  Our original sequence is:

    3
    4
    5
    2007-02-19-05:00
    11:19:24.734-05:00
    8
    blue
    red
    3.14
    42
    1995-04-21

  subsequence($testSequence, 8) =
    red
    3.14
    42
    1995-04-21

  subsequence($testSequence, 3, 3) =
    5
    2007-02-19-05:00
    11:19:24.734-05:00

  subsequence($testSequence, 1, 1) =
    3

  subsequence($testSequence, -1, 3) =
    3

  subsequence($testSequence, 14, 2) =


  subsequence($testSequence, 2.3, 1.7) =
    4
    5

  subsequence($emptySequence, 1) =
```

В первом примере функция начинает с восьмого объекта последовательности и возвращает все последующие объекты. Затем она начинает с третьего объекта и возвращает объекты 3, 4 и 5. Чтобы получить одиночный объект, мы используем параметры `1, 1` – первый объект, начиная с первого. Даже при передаче отрицательной начальной позиции вызов функции может возвращать объекты, как видно из примера

-1, 3. В данном случае запрашиваются три объекта в позициях -1, 0 и 1. Из них существует только объект в позиции 1, который и возвращается функцией subsequence(). В следующем примере с начальной позицией 14 функция возвращает пустую последовательность, потому что исходная последовательность содержит менее 14 объектов. Дробные позиции округляются до ближайшего целого числа, поэтому предпоследний вызов эквивалентен 2, 2. Наконец, вызов subsequence() для пустой последовательности возвращает пустую последовательность.

substring()

Возвращает часть исходной строки.

Синтаксис

```
[1.0] string substring($sourceString as string, $startingLoc as number,
                       $length as number?)
[2.0] xs:string substring($sourceString as xs:string?,
                          $startingLoc as xs:double)
[2.0] xs:string substring($sourceString as xs:string?,
                          $startingLoc as xs:double,
                          $length as xs:double)
```

Ввод

Функция substring() получает строку и одно или два числа. Строка содержит исходные данные, из которых выделяется подстрока. Второй аргумент определяет начальную позицию возвращаемой подстроки, а необязательный третий аргумент определяет количество возвращаемых символов.

Вывод

С двумя аргументами (строка и начальная позиция) функция substring() возвращает все символы строки, позиция которых больше либо равна заданной начальной позиции. *Учтите, что первый символ строки находится в позиции 1, а не в позиции 0.*

С тремя аргументами (строка, начальная позиция и длина) функция substring() возвращает все символы строки, позиция которых больше либо равна начальной, но меньше *суммы* начальной позиции и длины.

Обычно аргументы substring() представляют собой целые числа, хотя они могут быть также вещественными числами или более сложными выражениями. Вещественные числа преобразуются в целые функцией round().

Где определяется

[1.0] XPath, раздел 4.2 «String Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.4 «Functions on String Values».

Пример

Таблица стилей для демонстрации работы функции substring():

```
<?xml version="1.0"?>
<!-- substring.xsl -->
<xsl:stylesheet version="1.0"
```

```
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:output method="text"/>

    <xsl:template match="/">
      <xsl:text>&#xA;Tests of the substring() function:</xsl:text>

      <xsl:text>&#xA;&#xA;  1. substring('Now is the time', 4)     = "</xsl:text>
      <xsl:value-of select="substring('Now is the time', 4)"/>
      <xsl:text>"&#xA;</xsl:text>
      <xsl:text>  2. substring('Now is the time', 4, 6)  = "</xsl:text>
      <xsl:value-of select="substring('Now is the time', 4, 6)"/>
      <xsl:text>"&#xA;</xsl:text>

      <!-- Xalan не может обработать этот пример, Saxon и MSXSL могут. -->
      <xsl:text>  3. substring('Now is the time', 8, -6) = "</xsl:text>
      <xsl:value-of select="substring('Now is the time', 8, -6)"/>
      <xsl:text>"&#xA;</xsl:text>
      <xsl:text>  4. substring('Now is the time', -3, 6) = "</xsl:text>
      <xsl:value-of select="substring('Now is the time', -3, 6)"/>
      <xsl:text>"&#xA;</xsl:text>
      <xsl:text>  5. substring('Now is the time', 54, 6) = "</xsl:text>
      <xsl:value-of select="substring('Now is the time', 54, 6)"/>
      <xsl:text>"&#xA;&#xA;</xsl:text>
      <xsl:text>  6. substring('Now is the time', 1.7, 6.4) = "</xsl:text>
      <xsl:value-of select="substring('Now is the time', 1.7, 6.4)"/>
      <xsl:text>"&#xA;</xsl:text>
      <xsl:text>  7. substring('Now is the time', 1.7, 6.5) = "</xsl:text>
      <xsl:value-of select="substring('Now is the time', 1.7, 6.5)"/>
      <xsl:text>"&#xA;&#xA;</xsl:text>
      <xsl:text>  count(document(``)//*) = </xsl:text>
      <xsl:value-of select="count(document(``)//*)"/>
      <xsl:text>&#xA;&#xA;  8. substring('Here is a really, </xsl:text>
      <xsl:text>really, really long string',</xsl:text>
      <xsl:text>&#xA;     count(document(``)//*)) = "</xsl:text>
      <xsl:value-of
        select="substring('Here is a really, really, really long string',
                count(document(``)//*))"/>
      <xsl:text>"&#xA;</xsl:text>
    </xsl:template>

</xsl:stylesheet>
```

Процессоры Saxon и MSXSL выдают следующие результаты:

```
Tests of the substring() function:

  1. substring('Now is the time', 4) = " is the time"
  2. substring('Now is the time', 4, 6) = " is th"
  3. substring('Now is the time', 8, -6) = ""
  4. substring('Now is the time', -3, 6) = "No"
  5. substring('Now is the time', 54, 6) = ""

  6. substring('Now is the time', 1.7, 6.4) = "ow is "
  7. substring('Now is the time', 1.7, 6.5) = "ow is t"

  count(document(``)//*) = 32

  8. substring('Here is a really, really, really long string',
     count(document(``)//*)) = "y long string"
```

Рассмотрим каждый тестовый сценарий по отдельности:

1. Первый тест возвращает все символы с позицией `>= 4`.
2. Второй тест возвращает все символы с позицией `>= 4` и `< 10`. (Вспомните, что верхняя граница подстроки определяется суммированием начальной позиции и длины.)
3. Третий тест возвращает все символы с позицией `>= 8` и `< 2`. Конечно, этот бесполезный диапазон символов пуст.
4. Тест возвращает все символы с позицией `>= -3` и `< 3`. Казалось бы, подстрока с отрицательной начальной позицией не должна содержать символов, но позиции 1 и 2 попадают в указанный диапазон.
5. Тест не возвращает ничего, потому что начальная позиция больше длины строки.
6. В этом тесте начальная позиция и длина задаются вещественными числами. Два числа округляются, то есть вызов эквивалентен `substring('Now is the time', 2, 6)`. Возвращаются все символы с позициями `>= 2` и `< 8`.
7. В этом тесте тоже используются вещественные числа. В данном случае второе число (`6.5`) округляется до 7, поэтому вызов эквивалентен `substring('Now is the time', 2, 7)`. Возвращаются символы с позициями `>= 2` и `< 9`.
8. В двух последних случаях число генерируется более сложным выражением XPath (`count(document('')//*)`). Значение соответствует количеству элементов в самой таблице стилей. В данном тесте возвращаются все символы с позицией `>= 36`.

И последнее замечание: в Xalan пример `substring('Now is the time', 8, -6)` приводит к ошибке:

```
XSLT Error (javax.xml.transform.TransformerException):
java.lang.StringIndexOutOfBoundsException:
String index out of range: -6
```

Хотя эта ошибка допущена в реализации Xalan, она наглядно показывает, что вызов функции для некорректных данных может привести к неприятным последствиям.

substring-after()

Возвращает часть строки данных, следующую за первым вхождением искомой строки. Если искомая строка не найдена, функция `substring-after()` возвращает строку нулевой длины.

Синтаксис

```
[1.0] string substring-after(string, string)
[2.0] xs:string substring-after(xs:string?, xs:string?)
[2.0] xs:string substring-after(xs:string?, xs:string?,
                                $collation as xs:string)
```

Ввод

Две строки: строка данных, в которой осуществляется поиск, и искомая строка. Функция `substring-after()` ищет вхождение второй строки в строке данных.

[2.0] В XSLT 2.0 может передаваться необязательный третий аргумент: имя схемы упорядочения, определяющей правила сравнения строк.

Вывод

Часть строки данных, следующая за первым вхождением искомой строки. Если искомая строка не встречается в строке данных, функция возвращает пустую строку. Если вторая строка является строкой нулевой длины, `substring-after()` возвращает строку нулевой длины. Функция также возвращает строку нулевой длины, если первая строка короче второй.

Где определяется

[1.0] XPath, раздел 4.2 «String Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.5 «Functions Based on Substring Matching».

Пример

Таблица стилей с использованием функции `substring-after()`:

```
<?xml version="1.0"?>
<!-- substring-after1.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the substring-after() </xsl:text>
    <xsl:text>function:&#xA;&#xA;</xsl:text>

    <xsl:text>  substring-after('Abracadabra', </xsl:text>
    <xsl:text>'bra')  = "</xsl:text>
    <xsl:value-of
      select="substring-after('Abracadabra', 'bra')"/>
    <xsl:text>"&#xA;</xsl:text>

    <xsl:text>  substring-after('Abracadabra', </xsl:text>
    <xsl:text>'abra') = "</xsl:text>
    <xsl:value-of
      select="substring-after('Abracadabra', 'abra')"/>
    <xsl:text>"&#xA;</xsl:text>

    <xsl:text>  substring-after('Abracadabra', </xsl:text>
    <xsl:text>'A')    = "</xsl:text>
    <xsl:value-of
      select="substring-after('Abracadabra', 'A')"/>
    <xsl:text>"&#xA;</xsl:text>

    <xsl:text>  substring-after('Abracadabra', </xsl:text>
    <xsl:text>'a')    = "</xsl:text>
    <xsl:value-of
      select="substring-after('Abracadabra', 'a')"/>
    <xsl:text>"&#xA;</xsl:text>

    <xsl:text>  substring-after('Abracadabra', </xsl:text>
    <xsl:text>'')     = "</xsl:text>
    <xsl:value-of
      select="substring-after('Abracadabra', '')"/>
    <xsl:text>"&#xA;</xsl:text>
```

```
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
A test of the substring-after() function:

  substring-after('Abracadabra', 'bra')  = "cadabra"
  substring-after('Abracadabra', 'abra') = ""
  substring-after('Abracadabra', 'A')    = "bracadabra"
  substring-after('Abracadabra', 'a')    = "cadabra"
  substring-after('Abracadabra', '')     = "Abracadabra"
```

Во втором примере применяется таблица стилей XSLT 2.0 с пользовательской схемой упорядочения:

```
<?xml version="1.0"?>
<!-- substring-after2.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Another test of the substring-after() </xsl:text>
    <xsl:text>function:&#xA;&#xA;</xsl:text>

    <xsl:text>  substring-after('Schoenaicherstrasse', </xsl:text>
    <xsl:text>'Sch&#xF6;n') = &#xA;    "</xsl:text>
    <xsl:value-of
      select="substring-after('Schoenaicherstrasse', 'Sch&#xF6;n')"/>
    <xsl:text>"&#xA;</xsl:text>

    <xsl:text>  substring-after('Schoenaicherstrasse', </xsl:text>
    <xsl:text>'Sch&#xF6;n', [German collation]) = &#xA;    "</xsl:text>
    <xsl:value-of
      select="substring-after('Schoenaicherstrasse', 'Sch&#xF6;n',
              concat('http://saxon.sf.net/collation?',
                     'class=com.oreilly.xslt.GermanCollation;'))"/>
    <xsl:text>"&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Another test of the substring-after() function:

  substring-after('Schoenaicherstrasse', 'Schön') =
    ""
  substring-after('Schoenaicherstrasse', 'Schön', [German collation]) =
    "aicherstrasse"
```

Немецкая буква «о-умляут» (ö) эквивалентна oe. Упорядочение по умолчанию игнорирует это обстоятельство, но оно отражается в нашем пользовательском упорядочении. (Чтобы листинг поместился на странице, мы объединили две половины URI-адреса схемы упорядочения Saxon при помощи функции concat().) За дополнительной информацией обращайтесь к разделу «Функция document() и сортировка» главы 8.

substring-before()

Возвращает часть строки данных, предшествующую первому вхождению искомой строки. Если искомая строка не найдена, функция `substring-before()` возвращает строку нулевой длины.

Синтаксис

```
[1.0] string substring-before(string, string)
[2.0] xs:string substring-before(xs:string?, xs:string?)
[2.0] xs:string substring-before(xs:string?, xs:string?,
                                 $collation as xs:string)
```

Ввод

Две строки: строка данных, в которой осуществляется поиск, и искомая строка.

[2.0] В XSLT 2.0 может передаваться необязательный третий аргумент: имя схемы упорядочения, определяющей правила сравнения строк.

Вывод

Часть строки данных, предшествующая первому вхождению искомой строки. Если искомая строка не встречается в строке данных, функция возвращает пустую строку. Если вторая строка является строкой нулевой длины, `substring-before()` возвращает строку нулевой длины. Функция также возвращает строку нулевой длины, если первая строка короче второй.

Где определяется

[1.0] XPath, раздел 4.2 «String Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.5 «Functions Based on Substring Matching».

Пример

Таблица стилей с использованием функции `substring-before()`:

```
<?xml version="1.0"?>
<!-- substring-before1.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the substring-before() </xsl:text>
    <xsl:text>function:&#xA;&#xA;</xsl:text>

    <xsl:text>  substring-before('Abracadabra', </xsl:text>
    <xsl:text>'abra')        = "</xsl:text>
    <xsl:value-of
      select="substring-before('Abracadabra', 'abra')"/>
    <xsl:text>"&#xA;</xsl:text>

    <xsl:text>  substring-before('Abracadabra', </xsl:text>
    <xsl:text>'Abracadabra') = "</xsl:text>
    <xsl:value-of
```

```
      select="substring-before('Abracadabra', 'Abracadabra')"/>
    <xsl:text>&#xA;</xsl:text>

    <xsl:text>  substring-before('Abracadabra', </xsl:text>
    <xsl:text>'A')           = "</xsl:text>
    <xsl:value-of
      select="substring-before('Abracadabra', 'A')"/>
    <xsl:text>"&#xA;</xsl:text>

    <xsl:text>  substring-before('Abracadabra', </xsl:text>
    <xsl:text>'a')           = "</xsl:text>
    <xsl:value-of
      select="substring-before('Abracadabra', 'a')"/>
    <xsl:text>"&#xA;</xsl:text>

    <xsl:text>  substring-before('Abracadabra', </xsl:text>
    <xsl:text>'')            = "</xsl:text>
    <xsl:value-of
      select="substring-before('Abracadabra', '')"/>
    <xsl:text>"&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
A test of the substring-before() function:

  substring-before('Abracadabra', 'abra')        = "Abracad"
  substring-before('Abracadabra', 'Abracadabra') = ""
  substring-before('Abracadabra', 'A')           = ""
  substring-before('Abracadabra', 'a')           = "Abr"
  substring-before('Abracadabra', '')            = ""
```

Во втором примере используется таблица стилей XSLT 2.0 с пользовательской схемой упорядочения:

```
<?xml version="1.0"?>
<!-- substring-before2.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Another test of the substring-before() </xsl:text>
    <xsl:text>function:&#xA;&#xA;</xsl:text>

    <xsl:text>  substring-before('Schoenaicherstrasse', </xsl:text>
    <xsl:text>'stra&#xDF;e') = &#xA;    "</xsl:text>
    <xsl:value-of
      select="substring-before('Schoenaicherstrasse', 'stra&#xDF;e')"/>
    <xsl:text>"&#xA;</xsl:text>

    <xsl:text>  substring-before('Schoenaicherstrasse', </xsl:text>
    <xsl:text>'stra&#xDF;e', [German collation]) = &#xA;    "</xsl:text>
    <xsl:value-of
      select="substring-before('Schoenaicherstrasse', 'stra&#xDF;e',
              concat('http://saxon.sf.net/collation?',
                     'class=com.oreilly.xslt.GermanCollation;'))"/>
```

```
    <xsl:text>"&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Another test of the substring-before() function:

  substring-before('Schoenaicherstrasse', 'straße') =
    ""

  substring-before('Schoenaicherstrasse', 'straße', [German collation]) =
    "Schoenaicher"
```

В немецком языке символ «длинного s» (ß) эквивалентен ss. Схема упорядочения по умолчанию игнорирует это обстоятельство, но в нашей специализированной схеме оно учитывается. (Чтобы листинг поместился на странице, мы объединили две половины URI-адреса схемы упорядочения Saxon при помощи функции concat().) За дополнительной информацией обращайтесь к разделу «Функция document() и сортировка» главы 8.

sum()

Преобразует все узлы передаваемого набора в числа, а затем возвращает сумму всех полученных чисел.

Синтаксис

```
[1.0] number sum(node-set)
[2.0] xs:anyAtomicType sum(xs:anyAtomicType)
[2.0] xs:anyAtomicType sum(xs:anyAtomicType, $empty as xs:anyAtomicType)
```

Ввод

[1.0] Набор узлов. Любой узел в наборе узлов, не являющийся числом, преобразуется в число так, как если бы он был обработан функцией number(); после чего все числовые значения всех узлов суммируются.

[2.0] Последовательность атомарных значений плюс необязательное значение, возвращаемое в том случае, если первый аргумент sum() является пустой последовательностью.

Вывод

[1.0] Сумма числовых значений всех узлов в переданном наборе узлов. Если какой-либо узел в наборе не может быть преобразован в число, функция sum() возвращает NaN.

[2.0] Для заданной последовательности числовых значений sum() возвращает сумму значений. Для последовательности временных промежутков sum() возвращает сумму их продолжительностей. Как это часто бывает в XSLT 2.0, необходимо учитывать некоторые обстоятельства:

- Если сумма вычисляется для последовательности временных промежутков, все значения должны относиться к типу xs:dayTimeDurations или xs:yearMonthDurations. Смешанное использование двух типов недопустимо; если вы попытаетесь это сделать, процессор XSLT выдаст ошибку.

- Если какие-либо значения последовательности относятся к типу `xs:untypedAtomic`, процессор XSLT пытается преобразовать каждое значение к типу `xs:double`. Если какое-либо значение в последовательности не удается преобразовать к `xs:double`, процессор XSLT выдает ошибку.
- Если `sum()` передается непустая последовательность, а сумма равна `0`, функция возвращает `0`.
- При передаче `sum()` пустой последовательности по умолчанию возвращается `0`. Если по какой-либо причине вы хотите вернуть другое значение (например, строку `"Некорректный вызов"`), укажите необязательный аргумент `$empty`. Этот аргумент (если он задан) возвращается при вызове `sum()` для пустой последовательности.

Где определяется

[1.0] XPath, раздел 4.4 «Number Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.4 «Aggregate Functions».

Пример

Для демонстрации работы функции `sum()` используется следующий документ XML:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

Очень простая таблица стилей с использованием функции `sum()`:

```
<?xml version="1.0"?>
<!-- sum.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
```

```
    <xsl:text>&#xA;A test of the sum() function:&#xA;&#xA;</xsl:text>

    <xsl:text>  Total chocolate bar sales this quarter: </xsl:text>
    <xsl:value-of
      select="format-number(sum(/report/brand/units), '###,###')"/>
  </xsl:template>

</xsl:stylesheet>
```

Обработка документа XML с этой таблицей стилей дает следующий результат:

```
A test of the sum() function:

  Total chocolate bar sales this quarter: 91,316
```

Функция `format-number()` **используется для форматирования значения, возвращаемого функцией** `sum()`**.**

Следующая таблица стилей XSLT 2.0 вызывает функцию `sum()` **для нескольких разных последовательностей:**

```
<?xml version="1.0"?>
<!-- sum2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:variable name="seq1" select="(3, 5, 18)"/>
    <xsl:variable name="seq2" select="(3, 5, 48.273, 2.9e3)"/>

    <xsl:variable name="value1" as="xs:integer" select="42"/>
    <xsl:variable name="value2" as="xs:double" select="2718.28E-3"/>
    <xsl:variable name="value3" as="xs:float" select="98.6"/>
    <xsl:variable name="value4" as="xs:decimal" select="2.54"/>
    <xsl:variable name="seq3"
      select="($value1, $value2, $value3, $value4)"/>

    <xsl:variable name="seq4"
      select="(xs:yearMonthDuration('P3Y8M'),
               xs:yearMonthDuration('P4Y2M'),
               xs:yearMonthDuration('P6Y4M'))"/>
    <xsl:variable name="seq5"
      select="(xs:dayTimeDuration('P2DT4H23M12.2S'),
               xs:dayTimeDuration('P3DT8H17M'),
               xs:dayTimeDuration('P3D'))"/>

    <xsl:text>Here are some tests of the sum() function:&#xA;</xsl:text>

    <xsl:text>&#xA;  sum(</xsl:text>
    <xsl:value-of select="$seq1" separator=", "/>
    <xsl:text>) = </xsl:text>
    <xsl:value-of select="format-number(sum($seq1), '#.###')"/>

    <xsl:text>&#xA;&#xA;  sum(</xsl:text>
    <xsl:value-of select="$seq2" separator=", "/>
    <xsl:text>) = </xsl:text>
    <xsl:value-of select="format-number(sum($seq2), '#.###')"/>

    <xsl:text>&#xA;&#xA;  sum(</xsl:text>
```

```
<xsl:value-of select="$seq3" separator=", "/>
<xsl:text>) = </xsl:text>
<xsl:value-of select="format-number(sum($seq3), '#.###')"/>

<xsl:text>&#xA;&#xA;  sum(</xsl:text>
<xsl:value-of select="$seq4" separator=", "/>
<xsl:text>) = </xsl:text>
<xsl:value-of select="sum($seq4)"/>

<xsl:text>&#xA;&#xA;    In text, the sum of</xsl:text>
<xsl:for-each select="$seq4">
  <xsl:text>&#xA;      </xsl:text>
  <xsl:value-of select="years-from-duration(.)"/>
  <xsl:text> years and </xsl:text>
  <xsl:value-of select="months-from-duration(.)"/>
  <xsl:text> months (</xsl:text>
  <xsl:value-of select="."/>
  <xsl:text>)</xsl:text>
</xsl:for-each>

<xsl:text>&#xA;    is </xsl:text>
<xsl:value-of select="years-from-duration(sum($seq4))"/>
<xsl:text> years and </xsl:text>
<xsl:value-of select="months-from-duration(sum($seq4))"/>
<xsl:text> months (</xsl:text>
<xsl:value-of select="sum($seq4)"/>
<xsl:text>).</xsl:text>

<xsl:text>&#xA;&#xA;  sum(</xsl:text>
<xsl:value-of select="$seq5" separator=", "/>
<xsl:text>) = </xsl:text>
<xsl:variable name="sum5" select="sum($seq5)"/>
<xsl:value-of select="$sum5"/>

<xsl:text>&#xA;&#xA;    In text, the sum of</xsl:text>
<xsl:for-each select="$seq5">
  <xsl:text>&#xA;      </xsl:text>
  <xsl:value-of select="days-from-duration(.)"/>
  <xsl:text> days, </xsl:text>
  <xsl:value-of select="hours-from-duration(.)"/>
  <xsl:text> hours, </xsl:text>
  <xsl:value-of select="minutes-from-duration(.)"/>
  <xsl:text> minutes and </xsl:text>
  <xsl:value-of
    select="format-number(seconds-from-duration(.), '#.##')"/>
  <xsl:text> seconds (</xsl:text>
  <xsl:value-of select="."/>
  <xsl:text>)</xsl:text>
</xsl:for-each>

<xsl:text>&#xA;    is </xsl:text>
<xsl:value-of select="days-from-duration($sum5)"/>
<xsl:text> days, </xsl:text>
<xsl:value-of select="hours-from-duration($sum5)"/>
<xsl:text> hours, </xsl:text>
<xsl:value-of select="minutes-from-duration($sum5)"/>
<xsl:text> minutes and </xsl:text>
<xsl:value-of
  select="format-number(seconds-from-duration($sum5), '#.##')"/>
```

```
    <xsl:text> seconds.</xsl:text>

    <xsl:text>&#xA;&#xA;  sum(()) = </xsl:text>
    <xsl:value-of select="sum(())"/>

    <xsl:text>&#xA;&#xA;  sum((), 'Not applicable') = </xsl:text>
    <xsl:value-of select="sum((), 'Not applicable')"/>

    <xsl:text>&#xA;&#xA;  sum((3, -3), 'Not applicable') = </xsl:text>
    <xsl:value-of select="sum((3, -3), 'Not applicable')"/>

  </xsl:template>

</xsl:stylesheet>
```

Результат, полученный для таблицы стилей XSLT 2.0:

```
Here are some tests of the sum() function:

  sum(3, 5, 18) = 26

  sum(3, 5, 48.273, 2900) = 2956.273

  sum(42, 2.71828, 98.6, 2.54) = 145.858

  sum(P3Y8M, P4Y2M, P6Y4M) = P14Y2M

    In text, the sum of
      3 years and 8 months (P3Y8M)
      4 years and 2 months (P4Y2M)
      6 years and 4 months (P6Y4M)
    is 14 years and 2 months (P14Y2M).

  sum(P2DT4H23M12.2S, P3DT8H17M, P3D) = P8DT12H40M12.2S

    In text, the sum of
      2 days, 4 hours, 23 minutes and 12.2 seconds (P2DT4H23M12.2S)
      3 days, 8 hours, 17 minutes and 0 seconds (P3DT8H17M)
      3 days, 0 hours, 0 minutes and 0 seconds (P3D)
    is 8 days, 12 hours, 40 minutes and 12.2 seconds.

  sum(()) = 0

  sum((), 'Not applicable') = Not applicable

  sum((3, -3), 'Not applicable') = 0
```

system-property()

Возвращает значение системного свойства, имя которого задается аргументом функции.

Syntax

```
[1.0] object system-property(string)
[2.0] xs:string system-property(xs:string)
```

Ввод

Три свойства, определяемых спецификацией XSLT 1.0, должны поддерживаться всеми процессорами XSLT 1.0; процессоры XSLT 2.0 должны поддерживать пять дополнительных свойств, определяемых в спецификации XSLT 2.0. Конкретные про-

цессоры могут поддерживать и другие свойства, хотя разработчикам не разрешается добавлять новые свойства в пространство имен `xsl:`.

Все процессоры XSLT должны поддерживать три системных свойства:

`xsl:version`

Вещественное число, которое представляет версию XSLT, реализованную процессором XSLT. Свойство эквивалентно числовому значению `1.0`, хотя некоторые процессоры XSLT (Xalan, например) возвращают значение `1`. Процессоры XSLT 2.0 должны выдавать номер версии в виде вещественного числа (например, `2.0`).

`xsl:vendor`

Строка, идентифицирующая фирму-разработчика процессора XSLT.

`xsl:vendor-url`

Строка с URL-адресом фирмы-разработчика процессора XSLT. Обычно строка содержит адрес домашней страницы веб-сайта разработчика.

[2.0] Спецификация XSLT 2.0 определяет пять дополнительных свойств, которые должны поддерживаться всеми процессорами XSLT 2.0:

`xsl:product-name`

Имя процессора, заданное фирмой-разработчиком. Спецификация XSLT 2.0 рекомендует, чтобы имя оставалось неизменным при смене версии продукта, хотя это и не является обязательным требованием.

`xsl:product-version`

Версия процессора, определяемая фирмой-разработчиком. Спецификация XSLT 2.0 не предъявляет никаких требований к формату версии.

`xsl:is-schema-aware`

Строка `yes`, если процессор XSLT является схемосовместимым; строка `no` в противном случае.

`xsl:supports-serialization`

Строка `yes`, если процессор XSLT поддерживает сериализацию; строка `no` в противном случае.

`xsl:supports-backwards-compatibility`

Строка `yes`, если процессор XSLT обеспечивает обратную совместимость с более ранними версиями XSLT; строка `no` в противном случае.

Вывод

Значение запрашиваемого свойства.

Где определяется

[1.0] XSLT, раздел 12.4 «Miscellaneous Additional Functions».

[2.0] XSLT, раздел 16.6 «Miscellaneous Additional Functions».

Пример

Следующая таблица стилей запрашивает значения свойств, определяемых в XSLT 1.0:

```
<?xml version="1.0"?>
<!-- system-property1.xsl -->
<xsl:stylesheet version="1.0"
```

```
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the system-property() function:</xsl:text>
    <xsl:text>&#xA;&#xA;  xsl:version = "</xsl:text>
    <xsl:value-of select="system-property('xsl:version')"/>
    <xsl:text>"&#xA;</xsl:text>
    <xsl:text>  xsl:vendor = "</xsl:text>
    <xsl:value-of select="system-property('xsl:vendor')"/>
    <xsl:text>"&#xA;</xsl:text>
    <xsl:text>  xsl:vendor-url = "</xsl:text>
    <xsl:value-of select="system-property('xsl:vendor-url')"/>
    <xsl:text>"</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Результат обработки таблицы стилей процессором Saxon:

```
A test of the system-property() function:

  xsl:version = "2.0"
  xsl:vendor = "SAXON 9.0.0.3 from Saxonica"
  xsl:vendor-url = "http://www.saxonica.com/"
```

При обработке таблицы стилей процессором Xalan-J получается следующий результат:

```
A test of the system-property() function:

  xsl:version = "1"
  xsl:vendor = "Apache Software Foundation (Xalan XSLTC)"
  xsl:vendor-url = "http://xml.apache.org/xalan-j"
```

Процессор Microsoft XSLT выдает следующие значения:

```
A test of the system-property() function:

  xsl:version = "1"
  xsl:vendor = "Microsoft"
  xsl:vendor-url = "http://www.microsoft.com"
```

В последнем тесте ядро Altova XML используется в режиме XSLT 1.0:

```
A test of the system-property() function:

  xsl:version = "1"
  xsl:vendor = "Altova GmbH"
  xsl:vendor-url = "http://www.altova.com"
```

В режиме XSLT 2.0 процессор Altova возвращает строку `xsl:version = "2.0"`.

Следующая таблица стилей дополнительно тестирует свойства XSLT 2.0:

```
<?xml version="1.0"?>
<!-- system-property2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>
```

```
  <xsl:template match="/">
    <xsl:text>&#xA;A test of the system-property() function:</xsl:text>
    <xsl:text>&#xA;&#xA;  xsl:version = "</xsl:text>
    <xsl:value-of select="system-property('xsl:version')"/>
    <xsl:text>"&#xA;</xsl:text>
    <xsl:text>  xsl:vendor = "</xsl:text>
    <xsl:value-of select="system-property('xsl:vendor')"/>
    <xsl:text>"&#xA;</xsl:text>
    <xsl:text>  xsl:vendor-url = "</xsl:text>
    <xsl:value-of select="system-property('xsl:vendor-url')"/>
    <xsl:text>"&#xA;&#xA;XSLT 2.0 properties:&#xA;&#xA;</xsl:text>
    <xsl:text>  xsl:product-name = "</xsl:text>
    <xsl:value-of select="system-property('xsl:product-name')"/>
    <xsl:text>"&#xA;</xsl:text>
    <xsl:text>  xsl:product-version = "</xsl:text>
    <xsl:value-of select="system-property('xsl:product-version')"/>
    <xsl:text>"&#xA;</xsl:text>
    <xsl:text>  xsl:is-schema-aware = "</xsl:text>
    <xsl:value-of select="system-property('xsl:is-schema-aware')"/>
    <xsl:text>"&#xA;</xsl:text>
    <xsl:text>  xsl:supports-serialization = "</xsl:text>
    <xsl:value-of
      select="system-property('xsl:supports-serialization')"/>
    <xsl:text>"&#xA;</xsl:text>
    <xsl:text>  xsl:supports-backwards-compatibility = "</xsl:text>
    <xsl:value-of
      select="system-property('xsl:supports-backwards-compatibility')"/>
    <xsl:text>"</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

При обработке таблицы стилей в Saxon-B (процессоре, не являющемся схемосовместимым) результат выглядит так:

```
A test of the system-property() function:

  xsl:version = "2.0"
  xsl:vendor = "SAXON 9.0.0.3 from Saxonica"
  xsl:vendor-url = "http://www.saxonica.com/"

XSLT 2.0 properties:

  xsl:product-name = "SAXON"
  xsl:product-version = "9.0.0.3"
  xsl:is-schema-aware = "no"
  xsl:supports-serialization = "yes"
  xsl:supports-backwards-compatibility = "yes"
```

В схемосовместимом процессоре (Saxon-SA) та же таблица стилей выдает следующий результат:

```
A test of the system-property() function:

  xsl:version = "2.0"
  xsl:vendor = "SAXON 9.0.0.3 from Saxonica"
  xsl:vendor-url = "http://www.saxonica.com/"

XSLT 2.0 properties:
```

```
  xsl:product-name = "SAXON"
  xsl:product-version = "SA 9.0.0.3"
  xsl:is-schema-aware = "yes"
  xsl:supports-serialization = "yes"
  xsl:supports-backwards-compatibility = "yes"
```

Обработка таблицы стилей ядром Altova XML в режиме XSLT 2.0 дает следующий результат:

```
A test of the system-property() function:

  xsl:version = "2.0"
  xsl:vendor = "Altova GmbH"
  xsl:vendor-url = "http://www.altova.com"

XSLT 2.0 properties:

  xsl:product-name = "Altova XSLT Engine"
  xsl:product-version = "2007"
  xsl:is-schema-aware = "yes"
  xsl:supports-serialization = "yes"
  xsl:supports-backwards-compatibility = "no"
```

Наконец, процессоры XSLT могут поддерживать другие свойства. Например, Java-версия Saxon поддерживает системные свойства Java. Следующая таблица стилей запрашивает значения некоторых из этих свойств:

```
<?xml version="1.0"?>
<!-- system-property3.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Getting Java system properties with </xsl:text>
    <xsl:text>system-property():</xsl:text>
    <xsl:text>&#xA;&#xA;  java.version = "</xsl:text>
    <xsl:value-of select="system-property('java.version')"/>
    <xsl:text>"&#xA;  path.separator = "</xsl:text>
    <xsl:value-of select="system-property('path.separator')"/>
    <xsl:text>"&#xA;  file.separator = "</xsl:text>
    <xsl:value-of select="system-property('file.separator')"/>
    <xsl:text>"&#xA;  user.name = "</xsl:text>
    <xsl:value-of select="system-property('user.name')"/>
    <xsl:text>"&#xA;  user.country = "</xsl:text>
    <xsl:value-of select="system-property('user.country')"/>
    <xsl:text>"</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Результат, возвращаемый Saxon:

```
Getting Java system properties with system-property():

  java.version = "1.5.0_10"
  path.separator = ";"
  file.separator = "\"
  user.name = "Skippy"
  user.country = "US"
```

При обработке этой таблицы стилей в Altova для всех свойств Java выдаются пустые значения.

[2.0] timezone-from-date()

Возвращает компонент часового пояса для полученного значения `xs:date`.

Синтаксис

```
xs:dayTimeDuration? timezone-from-date(xs:date?)
```

Ввод

Значение `xs:date`.

Вывод

Значение `xs:dayTimeDuration`, представляющее компонент часового пояса в заданном значении `xs:date`. Если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Таблица стилей для демонстрации работы функции `timezone-from-date()`:

```
<?xml version="1.0"?>
<!-- timezone-from-date.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the timezone from an xs:date:</xsl:text>
    <xsl:variable name="currentDate" as="xs:date" select="current-date()"/>
    <xsl:text>&#xA;&#xA;  The current date is: </xsl:text>
    <xsl:value-of select="$currentDate"/>

    <xsl:text>&#xA;&#xA;  The current timezone is: </xsl:text>
    <xsl:value-of select="timezone-from-date($currentDate)"/>
    <xsl:text>&#xA;    The timezone is also known as </xsl:text>
    <xsl:value-of select="format-date($currentDate, '[ZN]')"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Extracting the timezone from an xs:date:

  The current date is: 2006-11-16-05:00

  The current timezone is: -PT5H
    The timezone is also known as EST
```

Обратите внимание: первый результат генерируется функцией `timezone-from-date()`, а второй – функцией `format-date()` с форматной строкой, выбирающей только компонент часового пояса из значения `xs:date`.

См. также

Определения функций *[2.0]* `day-from-date()`, *[2.0]* `format-date()`, *[2.0]* `month-from-date()` и *[2.0]* `year-from-date()`.

[2.0] timezone-from-dateTime()

Возвращает компонент часового пояса для полученного значения `xs:dateTime`.

Синтаксис

```
xs:dayTimeDuration? timezone-from-dateTime(xs:dateTime?)
```

Ввод

Значение `xs:dateTime`.

Вывод

Значение `xs:dayTimeDuration`, представляющее компонент часового пояса для заданного значения `xs:dateTime`. Если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Для демонстрации работы функции `timezone-from-dateTime()` используется следующая таблица стилей:

```
<?xml version="1.0"?>
<!-- timezone-from-datetime.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the timezone from an xs:dateTime:</xsl:text>
    <xsl:variable name="currentDateTime" as="xs:dateTime"
      select="current-dateTime()"/>
    <xsl:text>&#xA;&#xA;  The current date and time is: </xsl:text>
    <xsl:value-of select="$currentDateTime"/>

    <xsl:text>&#xA;&#xA;  The current timezone is: </xsl:text>
    <xsl:value-of select="timezone-from-dateTime($currentDateTime)"/>
    <xsl:text>&#xA;    The timezone is also known as </xsl:text>
    <xsl:value-of select="format-dateTime($currentDateTime, '[ZN]')"/>

  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Extracting the timezone from an xs:dateTime:

  The current date and time is: 2006-11-16T05:13:44.431-05:00

  The current timezone is: -PT5H
    The timezone is also known as EST
```

Обратите внимание: первый результат генерируется функцией `timezone-from-dateTime()`, а второй – функцией `format-dateTime()` с форматной строкой, выбирающей только компонент часового пояса из значения `xs:dateTime`.

См. также

Определения функций ***[2.0]*** `day-from-dateTime()`, ***[2.0]*** `format-dateTime()`, ***[2.0]*** `hours-from-dateTime()`, ***[2.0]*** `minutes-from-dateTime()`, ***[2.0]*** `month-from-dateTime()`, ***[2.0]*** `seconds-from-dateTime()` и ***[2.0]*** `year-from-dateTime()`.

[2.0] timezone-from-time()

Возвращает компонент часового пояса для полученного значения `xs:time`.

Синтаксис

```
xs:dayTimeDuration? timezone-from-time(xs:time?)
```

Ввод

Значение `xs:time`.

Вывод

Значение `xs:dayTimeDuration`, представляющее компонент часового пояса для заданного значения `xs:time`. Если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Таблица стилей для демонстрации работы функции `timezone-from-time()`:

```
<?xml version="1.0"?>
<!-- timezone-from-time.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the timezone from an xs:time:</xsl:text>
    <xsl:variable name="currentTime" as="xs:time" select="current-time()"/>
    <xsl:text>&#xA;&#xA;  The current time is: </xsl:text>
    <xsl:value-of select="$currentTime"/>
```

```
    <xsl:text>&#xA;&#xA;  The current timezone is: </xsl:text>
    <xsl:value-of select="timezone-from-time($currentTime)"/>
    <xsl:text>&#xA;    The timezone is also known as </xsl:text>
    <xsl:value-of select="format-time($currentTime, '[ZN]')"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Extracting the timezone component from an xs:time:

  The current time is: 05:15:30.423-05:00

  The current timezone is: -PT5H
    The timezone is also known as EST
```

Обратите внимание: первый результат генерируется функцией `timezone-from-time()`, а второй – функцией `format-time()` с форматной строкой, выбирающей только компонент часового пояса из значения `xs:time`.

См. также

Определения функций *[2.0]* `hours-from-time()`, *[2.0]* `format-time()`, *[2.0]* `minutes-from-dateTime()` и *[2.0]* `seconds-from-time()`.

[2.0] tokenize()

Разбивает строку на последовательность строк, используя регулярное выражение в качестве разделителя.

Синтаксис

```
xs:string* tokenize($input as xs:string?, $pattern as xs:string)
xs:string* tokenize($input as xs:string?, $pattern as xs:string,
                    $flags as xs:string)
```

Ввод

Строка для разбивки и регулярное выражение. Функция `tokenize()` может также получать необязательную строку с флагами, управляющими обработкой регулярного выражения.

Совпадение регулярного выражения со строкой нулевой длины является фатальной ошибкой. За подробностями обращайтесь к приложению E.

Вывод

Последовательность значений `xs:string`, каждое из которых представляет лексему, выделенную из исходной строки. Возвращаемые строки не содержат разделителя.

Полное описание правил работы `tokenize()`:

- Если в первом аргументе передается пустая последовательность или строка нулевой длины, функция возвращает пустую последовательность.
- Если регулярное выражение не совпадает в исходной строке, функция возвращает синглетную последовательность, содержащую исходную строку.

- Если регулярное выражение совпадает в начале строки, первая строка в возвращаемой последовательности будет пустой строкой (`""`). Аналогично, если регулярное выражение совпадает в конце строки, последняя строка в возвращаемой последовательности будет пустой строкой.
- Если регулярное выражение имеет два перекрывающихся совпадения в исходной строке, используется только первое совпадение.
- Регулярное выражение не может быть строкой нулевой длины и не может совпадать со строкой нулевой длины (другими словами, `matches("", $pattern, $replacement)` не может быть равно `true`). Если длина исходной строки отлична от нуля, сохраненная подстрока может иметь нулевую длину.

  Если выражение `matches("", $pattern, $replacement)` истинно, происходит ошибка. Например, при использовании шаблона `'.?'` с любой входной строкой происходит ошибка, потому что шаблон совпадает со строкой нулевой длины.
- При поиске совпадений для регулярных выражений не используются схемы упорядочения; сравниваются коды символов в Юникоде. Ситуации, в которых разные символы считаются одинаковыми в национальных языках, не учитываются.
- В отличие от регулярных выражений, используемых в элементе `<xsl:analyze-string>`, фигурные скобки (`{` и `}`) не удваиваются. (Фигурные скобки в атрибуте `regex` элемента `<xsl:analyze-string>` должны удваиваться, чтобы они не интерпретировались как шаблоны значений атрибутов.)
- Если регулярное выражение определяет более одной альтернативы и в некоторой позиции ввода совпадают сразу несколько из возможных альтернатив, используется первая альтернатива в регулярном выражении. Пример из спецификации: `tokenize("abracadabra", "(ab)|(a)")` возвращает (`""`, `"r"`, `"c"`, `"d"`, `"r"`, `""`).

  Первая альтернатива в регулярном выражении `(ab)` считается совпадением, несмотря на то, что вторая альтернатива `(a)` тоже совпадает.
- Наконец, параметр `$flags` управляет обработкой регулярного выражения. Поддерживаются четыре флага:

  `s` Регулярное выражение обрабатывается в режиме «совпадения точки со всеми символами». При наличии этого флага метасимвол «точка» (`.`) совпадает с любым символом. При стандартной обработке (без флага `s`) оператор «точка» совпадает с любым символом, *кроме* символа новой строки (`
`). Этот флаг особенно полезен при обработке строк, содержащих внутренние символы новой строки.

  `m` Регулярное выражение обрабатывается в многострочном режиме. В режиме по умолчанию метасимвол `^` совпадает с началом всей строки, а метасимвол `$` совпадает с концом каждой строки. В многострочном режиме `^` совпадает с началом, а `$` – с концом любой логической строки внутри «физической» строки.

  `i` Регулярное выражение обрабатывается без учета регистра символов. Регулярное выражение `"a"` совпадает как с `"a"`, так и с `"A"`.

  Учтите, что Юникод может создать массу проблем с этим флагом. Например, в спецификации XQuery 1.0 and XPath 2.0 Functions and Operators приводится пример с обозначением в Юникоде градусов по Кельвину (`K`), соответствующим букве `"K"`. Комбинация `regex="k"` и `flags="i"` совпадает как со знаком градусов по Кельвину, так и с буквами `"k"` (`k`) и `"K"` (`K`).

  Другие символы Юникода не преобразуются в буквы. Например, символ римской цифры `I` (`ࡰ`) внешне похож на букву `I`, но не преобразуется в нее. К счастью, эти проблемы выходят за рамки книги.

x Перед поиском совпадения из регулярного выражения удаляются все символы пропусков (	,
,
 и). Другими словами, с флагом x регулярные выражения "John Smith" и "JohnSmith" эквивалентны. Этот флаг особенно удобен при разбиении длинного регулярного выражения на несколько строк, чтобы его было легче читать.

Флаги могут комбинироваться в любом порядке. Наборы параметров 'xis' и 'six' работают абсолютно одинаково.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.6 «String Functions that Use Pattern Matching».

Пример

Таблица стилей для демонстрации работы функции tokenize():

```
<?xml version="1.0"?>
<!-- tokenize.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <xsl:variable name="string1"
      select="concat('Now is the time for all good men&#xA;',
              'and women &#xA;',
              'to aid the party.')"/>

    <xsl:variable name="string2" as="xs:string"
      select="'Visa # 1234-5678-9101-1121'"/>

    <xsl:text>&#xA;Here are some tests of the </xsl:text>
    <xsl:text>tokenize() function:&#xA;</xsl:text>

    <xsl:text>&#xA;  $string1 = &#xA;</xsl:text>
    <xsl:value-of select="$string1"/>

    <xsl:text>&#xA;&#xA;  $string2 = &#xA;</xsl:text>
    <xsl:value-of select="$string2"/>

    <xsl:text>&#xA;&#xA;  tokenize($string1, </xsl:text>
    <xsl:text>'&amp;#xA;') = </xsl:text>
    <xsl:text>&#xA;    ('</xsl:text>
    <xsl:value-of select="tokenize($string1, '&#xA;')"
      separator="',&#xA;      '"/>
    <xsl:text>')&#xA;    count() = </xsl:text>
    <xsl:value-of select="count(tokenize($string1, '&#xA;'))"/>

    <xsl:text>&#xA;&#xA;  tokenize($string2, '-') = </xsl:text>
    <xsl:text>&#xA;    ('</xsl:text>
    <xsl:value-of
      select="tokenize($string2, '-')" separator="', '"/>
    <xsl:text>')&#xA;    count() = </xsl:text>
    <xsl:value-of select="count(tokenize($string2, '-'))"/>

    <xsl:text>&#xA;&#xA;  tokenize($string2, </xsl:text>
```

```
    <xsl:text>'(Visa # )|-') = </xsl:text>
    <xsl:text>&#xA;    ('</xsl:text>
    <xsl:value-of
      select="tokenize($string2, '(Visa # )|-')" separator="', '"/>
    <xsl:text>')&#xA;    count() = </xsl:text>
    <xsl:value-of select="count(tokenize($string2, '(Visa #)|-'))"/>

  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here are some tests of the tokenize() function:

  $string1 =
Now is the time for all good men
and women
to aid the party.

  $string2 =
Visa # 1234-5678-9101-1121

  tokenize($string1, '&#xA;') =
    ('Now is the time for all good men',
     'and women ',
     'to aid the party.')
    count() = 3

  tokenize($string2, '-') =
    ('Visa # 1234', '5678', '9101', '1121')
    count() = 4

  tokenize($string2, '(Visa # )|-') =
    ('', '1234', '5678', '9101', '1121')
    count() = 5
```

В первом тесте строка разбивается по символам новой строки (`
`). Обратите внимание: в данном случае флаг `'s'` не используется, потому что мы ищем не символы в начале и конце строки, а просто символы новой строки. Во втором тесте номер кредитной карты разбивается на четыре фрагмента. К сожалению, первый фрагмент `'Visa # 1234'` содержит не ту часть строки, которая нам нужна. Чтобы выделить из строки только четыре числовых фрагмента, необходимо переписать регулярное выражение так, как это сделано в тесте 3. Так как шаблон `'Visa # '` совпадает в начале строки, первым фрагментом результата будет строка нулевой длины.

Функция `count()` возвращает количество объектов в последовательности.

См. также

В приложении E приведено подробное описание работы регулярных выражений в XPath 2.0. Обращайтесь также к определениям следующих элементов и функций: *[2.0]* `<xsl:analyze-string>`, *[2.0]* `matches()`, *[2.0]* `<xsl:matching-substring>`, *[2.0]* `<xsl:non-matching-substring>`, *[2.0]* `regex-group()` и *[2.0]* `replace()`.**[2.0] trace()**

[2.0] trace()

Выводит диагностическое сообщение, используемое в ходе трассировки таблицы стилей.

Синтаксис

```
item()* trace($value as item()*, $label as xs:string)
```

Ввод

Последовательность и значение `xs:string` (обычно строка имеет вид `The value of $x is:`). Диагностическое сообщение со строкой и значениями, объединенными в последовательность, может быть очень полезным в ходе трассировки таблицы стилей.

Вывод

Функция `trace()` выдает входную последовательность, которая возвращается в неизменном виде.

Кроме того, `trace()` преобразует последовательность объектов в `xs:string`. Обычно полученная строка и второй аргумент объединяются в диагностическое сообщение. *Почти все детали работы* `trace()` *зависят от реализации*. Формат диагностических сообщений, приемник для вывода диагностических сообщений, порядок обработки вызовов `trace()` – все это может изменяться в зависимости от процессора.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 4 «The Trace Function».

Пример

Следующая таблица стилей представляет собой сокращенную версию таблицы стилей, использовавшейся для демонстрации работы функции `data()`. Мы хотим, чтобы при каждом вызове пользовательской функции `datatest:print-item()` выводилось диагностическое сообщение. Таблица стилей выглядит так:

```
<?xml version="1.0"?>
<!-- trace.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:datatest="http://www.oreilly.com">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <xsl:variable name="testSequence" as="item()*">
      <xsl:sequence select="(3)"/>
      <xsl:element name="currentDate">
        <xsl:value-of select="current-date()"/>
      </xsl:element>
      <xsl:element name="currentTime">
        <xsl:value-of select="current-time()"/>
      </xsl:element>
      <xsl:element name="integerTest">
        <xsl:value-of select="xs:integer(8)"/>
      </xsl:element>
      <xsl:sequence select="('blue', 'red')"/>
      <xsl:element name="floatTest1">
        <xsl:value-of select="xs:float(3.14)"/>
      </xsl:element>
      <xsl:element name="floatTest2">
```

```
        <xsl:value-of select="xs:float(42)"/>
      </xsl:element>
      <xsl:element name="dateTest">
        <xsl:value-of select="xs:date('1995-04-21')"/>
      </xsl:element>
    </xsl:variable>

    <xsl:text>&#xA;Here is a test of the </xsl:text>
    <xsl:text>trace() function:</xsl:text>

    <xsl:value-of
      select="for $i in (1 to count($testSequence))
              return (datatest:print-item(
              subsequence($testSequence, $i, 1)))"/>
  </xsl:template>

  <xsl:function name="datatest:print-item" as="xs:string">
    <xsl:param name="item" as="item()"/>
    <xsl:choose>
      <xsl:when test="$item instance of element()">
        <xsl:analyze-string regex="!!!.?"
          select="string(trace($item, 'trace()  element '))">
          <xsl:matching-substring></xsl:matching-substring>
        </xsl:analyze-string>
      </xsl:when>
      <xsl:otherwise>
        <xsl:analyze-string regex="!!!.?"
          select="string(trace($item, 'trace()  atom '))">
          <xsl:matching-substring></xsl:matching-substring>
        </xsl:analyze-string>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:text> </xsl:text>
  </xsl:function>

</xsl:stylesheet>
```

Таблица стилей создает последовательность, а затем вызывает функцию `datatest:print-item()`**. При обработке таблицы процессором Saxon будет получен следующий результат:**

```
trace()  atom : xs:integer: 3
trace()  element : element(currentDate, untyped): currentDate
trace()  element : element(currentTime, untyped): currentTime
trace()  element : element(integerTest, untyped): integerTest
trace()  atom : xs:string: blue
trace()  atom : xs:string: red
trace()  element : element(floatTest1, untyped): floatTest1
trace()  element : element(floatTest2, untyped): floatTest2
trace()  element : element(dateTest, untyped): dateTest

Here is a test of the trace() function:

3
2006-12-19-05:00
17:13:55-05:00
8
blue
red
```

```
3.14
42
1995-04-21
```

Обратите внимание: все сообщения `trace()` выводятся до элементов `<xsl:text>` или `<xsl:value-of>`. Формат сообщений и порядок обработки вызовов `trace()` зависит от реализации. При обработке той же таблицы стилей ядром Altova XML будет получен несколько иной результат:

```
trace()  atom 3
trace()  element <currentDate>2006-12-19-05:00</currentDate>
trace()  element <currentTime>17:30:28-05:00</currentTime>
trace()  element <integerTest>8</integerTest>
trace()  atom blue
trace()  atom red
trace()  element <floatTest1>3.14</floatTest1>
trace()  element <floatTest2>42</floatTest2>
trace()  element <dateTest>1995-04-21</dateTest>

Here is a test of the trace() function:

3
...
```

И снова все диагностические сообщения выводятся до основных результатов.

translate()

Преобразует отдельные символы в строке, заменяя их другими значениями. Во многих языках эта функция обладает расширенными возможностями и может использоваться для преобразования регистра символов. (*[2.0]* Для преобразования регистра в XQuery 1.0 и XPath 2.0 существуют более мощные функции `lower-case()` и `upper-case()`.)

Синтаксис

```
[1.0] string translate(string, $mapString as string, $transString as string)
[2.0] xs:string translate(xs:string?, $mapString as xs:string,
                          $transString as xs:string)
```

Ввод

Три строки: исходная (непреобразованная) строка и две строки, определяющие преобразование символов.

Вывод

Исходная строка, преобразованная по следующим правилам:

- Если символ исходной строки присутствует во втором строковом аргументе (строка отображения), то он заменяется соответствующим символом третьего строкового аргумента (строка преобразования).

  Другими словами, вызов `translate('CAR', 'ABCDE', 'EXHQF')` возвращает `HER`. При последовательном переборе символов первой строки `C` находится на третьей позиции в строке отображения, поэтому этот символ заменяется `H` — третьим символом в строке преобразования. Следующий символ первой строки, `A`, является первым символом в строке отображения, поэтому он заменяется символом `E` — первым

символом в строке преобразования. Наконец, последний символ первой строки R не встречается во второй строке, поэтому он остается неизменным.

- Если символ исходной строки присутствует во втором строковом аргументе, но при этом не имеет соответствующего символа в третьем строковом аргументе (вторая строка длиннее третьей), этот символ удаляется.

  Другими словами, вызов translate('CAR', 'ABCDER', 'EXHQF') возвращает HE. Первые два символа преобразуются по правилам, описанным в предыдущем примере. Последняя буква исходной строки R является шестым символом строки отображения. Так как в строке преобразования шестого символа нет, символ удаляется.
- Если символ встречается во втором строковом аргументе несколько раз, символ замены определяется по первому вхождению.
- Если третий строковый аргумент длиннее второго, лишние символы игнорируются.
- Если вторая строка является строкой нулевой длины, исходная строка возвращается в неизменном виде.
- *[2.0]* Если значение первого аргумента является пустой последовательностью, функция возвращает строку нулевой длины.

Где определяется

[1.0] XPath, раздел 4.2 «String Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.4 «Functions on String Values».

Пример

Таблица стилей с несколькими примерами использования функции translate():

```
<?xml version="1.0"?>
<!-- translate.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Tests of the translate() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  Convert a string to uppercase:</xsl:text>
    <xsl:text>&#xA;    translate('Lily', </xsl:text>
    <xsl:text>'abcdefghijklmnopqrstuvwxyz', </xsl:text>
    <xsl:text>&#xA;               'ABCDEFGHIJKLMNOPQRSTUVWXYZ')=</xsl:text>
    <xsl:value-of
      select="translate('Lily', 'abcdefghijklmnopqrstuvwxyz',
                        'ABCDEFGHIJKLMNOPQRSTUVWXYZ')"/>
    <xsl:text>&#xA;&#xA;  Convert a string to lowercase:</xsl:text>
    <xsl:text>&#xA;    translate('Lily', </xsl:text>
    <xsl:text>'ABCDEFGHIJKLMNOPQRSTUVWXYZ', </xsl:text>
    <xsl:text>&#xA;               'abcdefghijklmnopqrstuvwxyz')=</xsl:text>
    <xsl:value-of
      select="translate('Lily', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
                        'abcdefghijklmnopqrstuvwxyz')"/>
    <xsl:text>&#xA;&#xA;  Remove parentheses, spaces, and dashes </xsl:text>
    <xsl:text>from a U.S. phone number:</xsl:text>
```

```
    <xsl:text>&#xA;    translate('(555) 555-1212', '() -', '')=</xsl:text>
    <xsl:value-of select="translate('(555) 555-1212', '() -', '')"/>
    <xsl:text>&#xA;&#xA;  Replace all but the last four digits of a </xsl:text>
    <xsl:text>credit card number with Xs:&#xA;</xsl:text>
    <xsl:variable name="credit" select="'1234 5678 9101 1810'"/>
    <xsl:text>    $credit='</xsl:text>
    <xsl:value-of select="$credit"/>
    <xsl:text>'</xsl:text>
    <xsl:text>&#xA;    translate(substring($credit, 1, 15), </xsl:text>
    <xsl:text>'1234567890 ', 'XXXXXXXXXX-')</xsl:text>
    <xsl:text>&#xA;    substring($credit, 16)</xsl:text>
    <xsl:text>&#xA;&#xA;    The first part is </xsl:text>
    <xsl:value-of
      select="translate(substring($credit, 1, 15), '1234567890 ',
                        'XXXXXXXXXX-')"/>
    <xsl:text>&#xA;    The second part is </xsl:text>
    <xsl:value-of select="substring($credit, 16)"/>
    <xsl:text>&#xA;    Here's how they look together: &#xA;      </xsl:text>
    <xsl:value-of
      select="translate(substring($credit, 1, 15), '1234567890 ',
                        'XXXXXXXXXX-')"/>
    <xsl:value-of select="substring($credit, 16)"/>
  </xsl:template>

</xsl:stylesheet>
```

При использовании с любым документом XML эта таблица стилей выводит следующий результат:

```
Tests of the translate() function:

  Convert a string to uppercase:
    translate('Lily', 'abcdefghijklmnopqrstuvwxyz',
              'ABCDEFGHIJKLMNOPQRSTUVWXYZ')=LILY

  Convert a string to lowercase:
    translate('Lily', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
              'abcdefghijklmnopqrstuvwxyz')=lily

  Remove parentheses, spaces, and dashes from a U.S. phone number:
    translate('(555) 555-1212', '() -', '')=5555551212

  Replace all but the last four digits of a credit card number with Xs:
    $credit='1234 5678 9101 1810'
    translate(substring($credit, 1, 15), '1234567890 ', 'XXXXXXXXXX-')
    substring($credit, 16)

    The first part is XXXX-XXXX-XXXX-
    The second part is 1810
    Here's how they look together:
      XXXX-XXXX-XXXX-1810
```

true()

Всегда возвращает логическое значение `true`. Вспомните, что строки «true» и «false» не имеют особой интерпретации в XSLT. Эта функция (вместе с функцией `false()`) позволяет генерировать логические значения там, где потребуется.

Синтаксис

```
[1.0] boolean true()
[2.0] xs:boolean true()
```

Ввод

Нет.

Вывод

Логическое значение `true`.

Где определяется

[1.0] XPath, раздел 4.3 «Boolean Functions».

[2.0] XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 9.1 «Additional Boolean Constructor Functions».

Пример

Короткий пример использования функции `true()`:

```
<?xml version="1.0"?>
<!-- true.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the true() function:&#xA;&#xA;</xsl:text>

    <xsl:text>  true() returned </xsl:text>
    <xsl:value-of select="true()"/>
    <xsl:text>!</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

Для любого документа XML эта таблица стилей выдает вполне предсказуемый результат:

```
A test of the true() function:

  true() returned true!
```

[2.0] type-available()

Для заданного типа данных возвращает `true`, если этот тип данных известен процессору XSLT.

Синтаксис

```
xs:boolean type-available($typeName as xs:string)
```

Ввод

Значение `xs:string`, содержащее имя типа данных.

Вывод

Значение `true` типа `xs:boolean`, если тип данных известен процессору XSLT; `false` в противном случае.

Где определяется

XSLT 2.0, раздел 18.1.4 «Testing Availability of Types».

Пример

Далее приводится короткая таблица стилей для проверки доступности двух типов данных. Первый тип данных определен для всех процессоров XSLT 2.0, а второй тип данных определяется в схеме, а следовательно, требует схемосовместимого процессора:

```
<?xml version="1.0"?>
<!-- type-available.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:age="http://www.oreilly.com/xslt"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the type-available() function:&#xA;</xsl:text>
    <xsl:text>&#xA;    xs:integer is available:  </xsl:text>
    <xsl:value-of select="type-available('xs:integer')"/>
    <xsl:text>&#xA;    age:age-type is available:  </xsl:text>
    <xsl:value-of select="type-available('age:age-type')"/>
  </xsl:template>

</xsl:stylesheet>
```

Обратите внимание на необходимость объявления префиксов пространств имен для типов данных. Элемент `<xsl:import-schema>` отсутствует в таблице стилей, поэтому доступен только первый тип данных:

```
A test of the type-available() function:

    xs:integer is available:  true
    age:age-type is available:  false
```

Чтобы другие типы стали доступными для схемосовместимого процессора, необходимо импортировать схему в таблицу стилей:

```
<?xml version="1.0"?>
<!-- type-available2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:age="http://www.oreilly.com/xslt"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:import-schema namespace="http://www.oreilly.com/xslt">
    <xs:schema
      targetNamespace="http://www.oreilly.com/xslt"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:simpleType name="age-type">
        <xs:restriction base="xs:integer">
```

```
          <xs:minInclusive value="0"/>
          <xs:maxInclusive value="130"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:schema>
  </xsl:import-schema>
  ...
</xsl:stylesheet>
```

После импортирования схемы таблица стилей сообщает о доступности второго типа данных:

```
A test of the type-available() function:

    xs:integer is available:  true
    age:age-type is available:  true
```

[2.0] unordered()

Возвращает объекты заданной последовательности в порядке, определяемом реализацией.

Синтаксис

```
item()* unordered(item()*)
```

Ввод

Последовательность объектов.

Вывод

Последовательность, которая содержит те же объекты, что и входная последовательность, но в порядке, определяемом процессором XSLT.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.1 «General Functions and Operators on Sequences».

Пример

Функция `unordered()` сообщает процессору XSLT, что порядок следования объектов в последовательности нас не интересует. В некоторых случаях это может повысить быстродействие процессора XSLT. В следующем примере мы проведем поиск предков элемента двумя способами. При стандартном решении предки следуют в порядке документа; при использовании `unorderer()` в Saxon предки следуют в обратном порядке. Скорее всего, Saxon использует свои внутренние структуры данных, чтобы начать поиск от узла и перечислять его предков, продвигаясь от внутреннего уровня документа наружу. При использовании той же таблицы стилей с AltovaXML предки в обоих случаях перечисляются в порядке документа.

Входные данные снова берутся из документа с описанием заказа:

```
<?xml version="1.0" ?>
<!-- po38293.xml -->
<purchase-order id="38293">
  <date year="2001" month="9" day="8"/>
  <customer id="4738" level="Basic">
```

```
    <address type="business">
      <name>
        <title>Ms.</title>
        <first-name>Amanda</first-name>
        <last-name>Reckonwith</last-name>
      </name>
      <street>930-A Chestnut Street</street>
      <city>Lynn</city>
      <state>MA</state>
      <zip>02930</zip>
    </address>
    <address type="ship-to"/>
  </customer>
  <items>
    <item part-no="23813-03-CDK">
      <name>Cucumber Decorating Kit</name>
      <qty>1</qty>
      <price>29.95</price>
    </item>
  </items>
</purchase-order>
```

Мы выведем список предков элемента `<first-name>`**, выбирая их дважды: обычным выражением XPath и функцией** `unordered()`**. Таблица стилей выглядит так:**

```
<?xml version="1.0"?>
<!-- unordered.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <xsl:text>&#xA;Here is a test of the unordered() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:for-each select="//first-name">
      <xsl:text>&#xA;  Element &lt;first-name&gt;</xsl:text>
      <xsl:value-of select="."/>
      <xsl:text>&lt;/first-name&gt;:&#xA;    </xsl:text>
      <xsl:text>&#xA;    Ancestors:&#xA;      </xsl:text>
      <xsl:value-of select="ancestor::*/name()" separator=", "/>
      <xsl:text>&#xA;&#xA;    Unordered ancestors:&#xA;      </xsl:text>
      <xsl:value-of select="unordered(ancestor::*/name())"
        separator=", "/>
    </xsl:for-each>

  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here is a test of the unordered() function:

  Element <first-name>Amanda</first-name>:

    Ancestors:
      purchase-order, customer, address, name
```

```
Unordered ancestors:
  name, address, customer, purchase-order
```

При первом способе выборки предки элемента следуют в порядке документа. При втором способе они следуют в другом порядке – вероятно, в том, который, по мнению Saxon, обеспечивает наибольшую эффективность поиска.

[2.0] unparsed-entity-public-id()

Возвращает открытый идентификатор неразобранной сущности с заданным именем. Если такой сущности нет, `unparsed-entity-public-id()` возвращает пустую строку.

Синтаксис

```
xs:string unparsed-entity-public-id(xs:string)
```

Ввод

Имя неразобранной сущности.

Вывод

Общедоступный идентификатор неразобранной сущности с заданным именем.

Где определяется

XSLT 2.0, раздел 16.6 «Miscellaneous Additional Functions».

Пример

Неразобранные сущности редко применяются на практике; обычно они представляют данные в формате, отличном от XML, как, например, сущность `author-picture`, определяемая в следующем документе XML:

```
<?xml version="1.0"?>
<!-- unparsed-entity.xml -->
<!DOCTYPE book [
  <!ENTITY author-picture PUBLIC "-//OReilly//Author Images//DT"
    "dougtidwell.jpg" NDATA JPEG>
]>

<book>
  <prolog cover-image="author-picture"/>
  <body>
    <p>Pretend that lots of useful content appears here.</p>
  </body>
</book>
```

Для получения открытого идентификатора неразобранной сущности используется следующая таблица стилей:

```
<?xml version="1.0"?>
<!-- unparsed-entity-public-id.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
```

```
    <xsl:text>&#xA;A test of the unparsed-entity-</xsl:text>
    <xsl:text>public-id() function:</xsl:text>

    <xsl:text>&#xA;&#xA;   </xsl:text>

    <xsl:text>The public ID of the cover image is:&#xA;      </xsl:text>
    <xsl:value-of
      select="unparsed-entity-public-id(/book/prolog/@cover-image)"/>
    <xsl:text>.&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Результат преобразования этого документа XML по таблице стилей выглядит так:

```
A test of the unparsed-entity-public-id() function:

   The public ID of the cover image is:
      -//OReilly//Author Images//DT.
```

unparsed-entity-uri()

Возвращает URI-адрес неразобранной сущности с заданным именем. Если такой сущности нет, `unparsed-entity-uri()` возвращает пустую строку.

Синтаксис

```
[1.0] string unparsed-entity-uri(string)
[2.0] xs:anyURI unparsed-entity-uri(xs:string)
```

Ввод

Имя неразобранной сущности.

Вывод

URI-адрес неразобранной сущности с заданным именем.

Где определяется

[1.0] XSLT, раздел 12.4 «Miscellaneous Additional Functions».

[2.0] XSLT, раздел 16.6 «Miscellaneous Additional Functions».

Пример

Неразобранные сущности редко применяются на практике; обычно они представляют данные в формате, отличном от XML, как, например, сущность `author-picture`, определяемая в следующем документе XML:

```
<?xml version="1.0"?>
<!-- unparsed-entity.xml -->
<!DOCTYPE book [
  <!ENTITY author-picture PUBLIC "-//OReilly//Author Images//DT"
    "dougtidwell.jpg" NDATA JPEG>
]>

<book>
  <prolog cover-image="author-picture"/>
  <body>
    <p>Pretend that lots of useful content appears here.</p>
```

```
  </body>
</book>
```

Для получения URI-адреса неразобранной сущности используется следующая таблица стилей:

```
<?xml version="1.0"?>
<!-- unparsed-entity-uri.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;A test of the unparsed-entity-</xsl:text>
    <xsl:text>uri() function:</xsl:text>

    <xsl:text>&#xA;&#xA;   </xsl:text>
    <xsl:text>The URI of the cover image is:&#xA;      </xsl:text>
    <xsl:value-of
      select="unparsed-entity-uri(/book/prolog/@cover-image)"/>
    <xsl:text>.&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Преобразование документа XML по таблице стилей дает следующий результат:

```
A test of the unparsed-entity-uri() function:

   The URI of the cover image is:
      file:/C:/projects/XSLTbookV2/AppendixC/dougtidwell.jpg.
```

URI-адрес неразобранной сущности основан на базовом URI-адресе самого документа XML.

[2.0] unparsed-text()

По заданному URI-адресу возвращает неразобранный текст ресурса, идентифицируемого этим URI-адресом.

Синтаксис

```
xs:string unparsed-text($href as xs:string)
xs:string unparsed-text($href as xs:string, $encoding as xs:string)
```

Ввод

Значение `xs:string`, определяющее URI-адрес запрашиваемого документа. Необязательный второй аргумент задает кодировку документа.

Вывод

Значение `xs:string`, содержащее неразобранный текст ресурса.

Где определяется

XSLT 2.0, раздел 16.2 «Reading Text Files»,

Пример

В этом примере генерируется страница HTML. Часть страницы содержит стандартный заголовок и завершитель; они вставляются в элемент в виде неразобранного текста. Если бы мы использовали функцию `collection()`, `doc()` или `document()`, читаемые данные должны были бы представлять собой корректный (well-formed) код XML. Функция `unparsed-text()` позволяет загружать и использовать документы, не содержащие корректного кода XML (как, например, большинство документов HTML).

Для получения исходных данных мы воспользуемся следующим документом XML:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- chocolate.xml -->
<report month="8" year="2006">
  <title>Chocolate bar sales</title>
  <brand>
    <name>Lindt</name>
    <units>27408</units>
  </brand>
  <brand>
    <name>Callebaut</name>
    <units>8203</units>
  </brand>
  <brand>
    <name>Valrhona</name>
    <units>22101</units>
  </brand>
  <brand>
    <name>Perugina</name>
    <units>14336</units>
  </brand>
  <brand>
    <name>Ghirardelli</name>
    <units>19268</units>
  </brand>
</report>
```

На базе этого документа мы сгенерируем отчет в формате HTML. В таблице стилей функция `unparsed-text()` импортирует заголовок и завершитель HTML:

```
<?xml version="1.0"?>
<!-- unparsed-text.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:variable name="reportTitle">
    <xsl:value-of select="/report/title"/>
    <xsl:text> for </xsl:text>
    <xsl:value-of select="/report/@month"/>
    <xsl:text>/</xsl:text>
    <xsl:value-of select="/report/@year"/>
  </xsl:variable>

  <xsl:template match="/">
    <html>
      <head>
        <title>
```

```
          <xsl:value-of select="$reportTitle"/>
        </title>
      </head>
      <body style="font-family: sans-serif;">
        <xsl:value-of
          select="unparsed-text('header.html')"
          disable-output-escaping="yes"/>
        <xsl:apply-templates select="*|text()"/>
        <xsl:value-of
          select="unparsed-text('footer.html')"
          disable-output-escaping="yes"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="title">
    <h1>
      <xsl:value-of select="$reportTitle"/>
    </h1>
  </xsl:template>

  <xsl:template match="brand">
    <h2>
      <xsl:value-of select="name"/>
      <xsl:text> : </xsl:text>
      <xsl:value-of select="units"/>
    </h2>
  </xsl:template>

</xsl:stylesheet>
```

Обратите внимание на использование атрибута `disable-output-escaping` в элементе `<xsl:value-of>`. Без этого атрибута импортируемая разметка будет отображаться в виде `<td>This header was generated...`. Он означает, что импортируемый текст должен интерпретироваться именно как текст, а не как разметка, предназначенная для обработки броузером.

Документ *header.html* выглядит так:

```
<!-- header.html -->
<table width=33% border=1>
  <tr style="background: lightgray;">
    <td style="text-align: center;
               font-weight: bold;">
      This header was generated by<br>
      <code>unparsed-text()</code>, using a
      stylesheet from <br>O'Reilly's
      <cite>XSLT</cite>, 2nd edition.
    </td>
  </tr>
</table>
```

Обратите внимание: в элементах `<table>` и `<br>` документа HTML встречаются атрибуты, значения которых не заключены в кавычки. Несмотря на нарушения синтаксиса XML, мы все равно можем импортировать документ и использовать его в сгенерированных результатах.

(Файл *footer.html* выглядит точно так же, не считая изменившегося названия.)

На рис. C.6 показано, как выглядит итоговый документ.

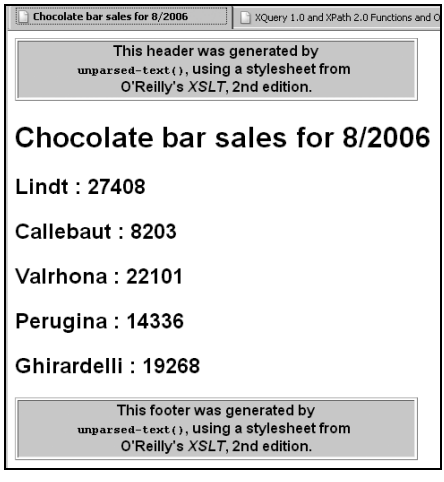


***Рис. C.6.** Файл HTML, созданный с использованием функции unparsed-text( )*

Функция `unparsed-text()` также хорошо подходит для чтения структурированных форматов данных, например данных, разделенных запятыми. За подробным примером обращайтесь к разделу «[2.0] Функции unparsed-text() и unparsed-text-available()» главы 8.

[2.0] unparsed-text-available()

Для заданного URI-адреса возвращает `true`, если документ доступен, и `false` в противном случае. Благодаря этой функции вы можете проверить существование и доступность документа, прежде чем открывать его функцией `unparsed-text()`. При попытке открытия недоступного документа процессор XSLT выдает фатальную ошибку.

Синтаксис

```
xs:boolean unparsed-text-available(xs:string)
xs:boolean unparsed-text-available(xs:string, xs:string)
```

Ввод

Значение `xs:string`, определяющее URI-адрес запрашиваемого документа. Необязательный второй аргумент определяет кодировку документа.

Вывод

`true`, если документ доступен; `false` в противном случае.

Где определяется

XSLT 2.0, раздел 16.2 «Reading Text Files».

Пример

Короткая таблица стилей проверяет доступность трех разных файлов:

```
<?xml version="1.0"?>
<!-- unparsed-text-available.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <xsl:text>&#xA;Here are some tests of the </xsl:text>
    <xsl:text>unparsed-text-available() function:</xsl:text>

    <xsl:text>&#xA;&#xA;  unparsed-text-available</xsl:text>
    <xsl:text>('header.html') = </xsl:text>
    <xsl:value-of
      select="unparsed-text-available('header.html')"/>

    <xsl:text>&#xA;&#xA;  unparsed-text-available</xsl:text>
    <xsl:text>('disclaimer.html') = </xsl:text>
    <xsl:value-of
      select="unparsed-text-available('disclaimer.html')"/>

    <xsl:text>&#xA;&#xA;  unparsed-text-available</xsl:text>
    <xsl:text>('footer.html') = </xsl:text>
    <xsl:value-of
      select="unparsed-text-available('footer.html')"/>

  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Here are some tests of the unparsed-text-available() function:

  unparsed-text-available('header.html') = true

  unparsed-text-available('disclaimer.html') = false

  unparsed-text-available('footer.html') = true
```

Второй файл недоступен, а первый и третий доступны.

[2.0] upper-case()

Преобразует строку к верхнему регистру.

Синтаксис

```
xs:string upper-case(xs:string?)
```

Ввод

Значение `xs:string`.

Вывод

Значение `xs:string`, в котором все строчные буквы оригинала преобразованы в прописные. Все символы, изначально записанные в верхнем регистре, а также символы,

не имеющие отдельной разновидности в верхнем регистре, возвращаются в неизменном виде. Если значение аргумента является пустой строкой, функция возвращает строку нулевой длины.

Из-за диакритических знаков и других особенностей национальных языков при приведении к верхнему регистру длина строки может измениться. Также следует учитывать, что в некоторых языках функции `lower-case()` и `upper-case()` не являются взаимно противоположными. Все правила преобразования регистра определяются в стандарте Юникода и должны соблюдаться процессорами XSLT.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.4 «Functions on String Values».

Пример

Следующая таблица стилей демонстрирует использование функции `upper-case()`. Обратите внимание на элемент `<xsl:output method="xml" encoding="UTF-8"/>`, обеспечивающий правильную обработку кодировки символов:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- upper-case.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" encoding="UTF-8" indent="yes"/>

  <xsl:template match="/">
    <testcase>
      <heading>Tests of the upper-case() function:</heading>
      <test>
        <label>upper-case('Lily') = </label>
        <result><xsl:value-of select="upper-case('Lily')"/></result>
      </test>
      <test>
        <label>upper-case('LILY') = </label>
        <result><xsl:value-of select="upper-case('LILY')"/></result>
      </test>
      <test>
        <label>upper-case('lily') = </label>
        <result><xsl:value-of select="upper-case('lily')"/></result>
      </test>
      <test>
        <label>uppercase('jalapeco') = </label>
        <result><xsl:value-of select="upper-case('jalapeco')"/></result>
      </test>
    </testcase>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
<?xml version="1.0" encoding="UTF-8"?>
<testcase>
   <heading>Tests of the upper-case() function:</heading>
   <test>
      <label>upper-case('Lily') = </label>
```

```
      <result>LILY</result>
   </test>
   <test>
      <label>upper-case('LILY') = </label>
      <result>LILY</result>
   </test>
   <test>
      <label>upper-case('lily') = </label>
      <result>LILY</result>
   </test>
   <test>
      <label>uppercase('jalapeco') = </label>
      <result>JALAPECO</result>
   </test>
</testcase>
```

[2.0] year-from-date()

Возвращает компонент года для полученного значения `xs:date`.

Синтаксис

```
xs:integer? year-from-date(xs:date?)
```

Ввод

Значение `xs:date`.

Вывод

Значение `xs:integer`, представляющее компонент года для заданного значения `xs:date`. Если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Для демонстрации работы функции `year-from-date()` используется следующая таблица стилей:

```
<?xml version="1.0"?>
<!-- year-from-date.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the year from an xs:date:</xsl:text>
    <xsl:variable name="currentDate" as="xs:date" select="current-date()"/>
    <xsl:text>&#xA;&#xA;  The current date is: </xsl:text>
    <xsl:value-of select="$currentDate"/>

    <xsl:text>&#xA;&#xA;  The current year: </xsl:text>
```

```
    <xsl:value-of select="year-from-date($currentDate)"/>
    <xsl:text>&#xA;    In words: </xsl:text>
    <xsl:value-of select="format-date($currentDate, '[YWw]')"/>
    <xsl:text>&#xA;    In German: </xsl:text>
    <xsl:value-of select="format-date($currentDate, '[YWw]', 'de', (), ())"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Extracting the year from an xs:date:

  The current date is: 2006-11-16-05:00

  The current year: 2006
    In words: Two Thousand and Six
    In German: Zweitausend Sechs
```

См. также

Определения функций *[2.0]* `day-from-date()`, *[2.0]* `format-date()`, *[2.0]* `month-from-date()` и *[2.0]* `timezone-from-date()`.

[2.0] year-from-dateTime()

Возвращает компонент года для полученного значения `xs:dateTime`.

Синтаксис

```
xs:integer? year-from-dateTime(xs:dateTime?)
```

Ввод

Значение `xs:dateTime`.

Вывод

Значение `xs:integer`, представляющее компонент года для заданного значения `xs:dateTime`. Если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Для демонстрации работы функции `year-from-dateTime()` используется следующая таблица стилей:

```
<?xml version="1.0"?>
<!-- year-from-datetime.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
```

```
    <xsl:text>&#xA;Extracting the year from an xs:dateTime:</xsl:text>
    <xsl:variable name="currentDateTime" as="xs:dateTime"
      select="current-dateTime()"/>
    <xsl:text>&#xA;&#xA;The current date and time is: </xsl:text>
    <xsl:value-of select="$currentDateTime"/>

    <xsl:text>&#xA;&#xA;  The current year: </xsl:text>
    <xsl:value-of select="year-from-dateTime($currentDateTime)"/>
    <xsl:text>&#xA;    In words: </xsl:text>
    <xsl:value-of select="format-dateTime($currentDateTime, '[YWw]')"/>
    <xsl:text>&#xA;    In German: </xsl:text>
    <xsl:value-of
      select="format-dateTime($currentDateTime, '[YWw]', 'de', (), ())"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей создает следующий результат:

```
The current date and time is: 2006-11-16T05:18:20.187-05:00

  The current year: 2006
    In words: Two Thousand and Six
    In German: Zweitausend Sechs
```

См. также

Определения функций *[2.0]* `day-from-dateTime()`, *[2.0]* `format-dateTime()`, *[2.0]* `hours-from-dateTime()`, *[2.0]* `minutes-from-dateTime()`, *[2.0]* `month-from-dateTime()`, *[2.0]* `seconds-from-dateTime()` и *[2.0]* `timezone-from-dateTime()`.

[2.0] years-from-duration()

Возвращает количество лет в заданном временном интервале `xs:duration`.

Синтаксис

```
xs:integer? years-from-duration(xs:duration?)
```

Ввод

Значение `xs:duration`.

Вывод

Значение `xs:integer`, представляющее количество лет в заданном временном интервале `xs:duration`. Учтите, что для `xs:dayTimeDuration` эта функция всегда возвращает `0`, потому что в значении `xs:dayTimeDuration` отсутствует компонент годов. Кроме того, если аргумент является пустой последовательностью, функция возвращает пустую последовательность.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 10.5 «Component Extraction Functions on Durations, Dates and Times».

Пример

Следующая таблица стилей демонстрирует использование функции years-from-duration() для всех трех типов временных промежутков:

```
<?xml version="1.0"?>
<!-- years-from-duration.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Extracting the years component from durations:</xsl:text>

    <xsl:variable name="sampleDuration" as="xs:duration"
      select="xs:duration('P3Y8M2DT4H23M12.2S')"/>
    <xsl:variable name="sampleYearMonthDuration" as="xs:yearMonthDuration"
      select="xs:yearMonthDuration('P3Y8M')"/>
    <xsl:variable name="sampleDayTimeDuration" as="xs:dayTimeDuration"
      select="xs:dayTimeDuration('P2DT4H23M12.2S')"/>

    <xsl:text>&#xA;&#xA;  A sample xs:duration: </xsl:text>
    <xsl:value-of select="$sampleDuration"/>
    <xsl:text>&#xA;    The years component of this duration is </xsl:text>
    <xsl:value-of select="years-from-duration($sampleDuration)"/>
    <xsl:text>.</xsl:text>

    <xsl:text>&#xA;&#xA;  A sample xs:yearMonthDuration: </xsl:text>
    <xsl:value-of select="$sampleYearMonthDuration"/>
    <xsl:text>&#xA;    The years component of this duration is </xsl:text>
    <xsl:value-of select="years-from-duration($sampleYearMonthDuration)"/>
    <xsl:text>.</xsl:text>

    <xsl:text>&#xA;&#xA;  A sample xs:dayTimeDuration: </xsl:text>
    <xsl:value-of select="$sampleDayTimeDuration"/>
    <xsl:text>&#xA;    The years component of this duration is </xsl:text>
    <xsl:value-of select="years-from-duration($sampleDayTimeDuration)"/>
    <xsl:text>.</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Extracting the years component from durations:

  A sample xs:duration: P3Y8M2DT4H23M12.2S
    The years component of this duration is 3.

  A sample xs:yearMonthDuration: P3Y8M
    The years component of this duration is 3.

  A sample xs:dayTimeDuration: P2DT4H23M12.2S
    The years component of this duration is 0.
```

Как упоминалось ранее, при вызове years-from-duration() для xs:dayTimeDuration возвращается 0.

См. также

Определения функций *[2.0]* days-from-duration(), *[2.0]* hours-from-duration(), *[2.0]* minutes-from-duration(), *[2.0]* months-from-duration() и *[2.0]* seconds-from-duration().

[2.0] zero-or-one()

Выдает ошибку, если аргумент не является последовательностью, содержащей нуль или один объект. Например, если элемент <name> может содержать не более одного дочернего элемента <title>, вызов zero-or-one(title) выдает ошибку, если элемент <name> содержит недопустимое количество дочерних элементов <title>. Учтите, что zero-or-one() завершает обработку таблицы стилей. Если вам потребуется более гибкий подход, используйте функцию count() для определения количества объектов в последовательности.

Синтаксис

```
item()? zero-or-one(item()*)
```

Ввод

Последовательность (которая должна быть пустой или синглетной).

Вывод

Если входная последовательность содержит нуль или один объект, функция возвращает входную последовательность. В противном случае zero-or-one() выдает ошибку.

Где определяется

XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 15.2 «Functions That Test the Cardinality of Sequences». Дополнительная информация об этой функции приводится в XQuery 1.0 and XPath 2.0 Formal Semantics, раздел 7.2 «Standard Functions with Specific Static Typing Rules».

Пример

Рассмотрим тривиальный пример, демонстрирующий использование функции zero-or-one(). Вызов zero-or-one() для последовательности, содержащей более одного элемента, выдает ошибку. Таблица стилей:

```
<?xml version="1.0"?>
<!-- zero-or-one.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text"/>

  <xsl:template match="/">

    <xsl:variable name="emptySequence" as="item()*">
      <xsl:sequence select="()"/>
    </xsl:variable>

    <xsl:variable name="singleton" as="item()*">
      <xsl:sequence select="(3)"/>
    </xsl:variable>

    <xsl:text>&#xA;Here are two tests of the zero-or-one() </xsl:text>
    <xsl:text>function:&#xA;</xsl:text>

    <xsl:text>&#xA;  Calling zero-or-one() with the empty </xsl:text>
    <xsl:text>sequence:&#xA;    </xsl:text>
```

```
    <xsl:if test="count(zero-or-one($emptySequence) &lt; 2)">
      <xsl:text>&#xA;    Our sequence has zero or one items!</xsl:text>
    </xsl:if>

    <xsl:text>&#xA;&#xA;  Calling zero-or-one() with a </xsl:text>
    <xsl:text>singleton:&#xA;</xsl:text>
    <xsl:text>&#xA;    Our sequence is:&#xA;      </xsl:text>
    <xsl:value-of select="$singleton" separator="&#xA;    "/>

    <xsl:if test="count(zero-or-one($singleton) &lt; 2)">
      <xsl:text>&#xA;&#xA;    Our sequence has zero or one items!</xsl:text>
    </xsl:if>
  </xsl:template>

</xsl:stylesheet>
```

Обратите внимание на использование функции `count()` для создания логического значения. Прямое преобразование последовательности в логическое значение здесь не работает. Если последовательность является пустой, процессор XSLT 2.0 интерпретирует ее как `false`, а это совсем не то, что нам требуется. Если последовательность состоит из одного объекта, проблемы все равно остаются; последовательность `(0)` интерпретируется как `false`, а последовательность `(xs:date('1995-04-21'))` приводит к ошибке. Чтобы получить здесь логическое значение, лучше всего проверить, действительно ли размер возвращаемой последовательности `< 2`.

Таблица стилей выдает следующий результат:

```
Here are two tests of the zero-or-one() function:

  Calling zero-or-one() with the empty sequence:

    Our sequence has zero or one items!

  Calling zero-or-one() with a singleton:

    Our sequence is:
      3

    Our sequence has zero or one items!
```

Вызов `zero-or-one()` для пустой или синглетной последовательности возвращает входную последовательность; вызов `zero-or-one()` для чего-либо другого выдает ошибку.

См. также

Описания функций `count()`, *[2.0]* `empty()`, *[2.0]* `exactly-one()` и *[2.0]* `one-or-more()`, описание оператора XPath 2.0 `treat as` (раздел «[2.0] Операторы типов данных – instance of, castable as, cast as и treat as» главы 3).

D

Краткий обзор XML Schema

В этом приложении приводится краткий обзор XML Schema. Если вы хотите подробнее ознакомиться с XML Schema, рекомендую книгу Эрика ван дер Влиста (Eric van der Vlist) «XML Schema» (O'Reilly). Документ «XML Schema Primer» на сайте W3C тоже содержит очень полезную информацию.

В приложении обсуждаются три темы. Сначала показывается, как в XML Schema объявляются элементы и атрибуты. Далее рассматриваются многочисленные способы создания типов. В завершение демонстрируется, как использовать XML Schema в таблицах стилей XSLT. Основное внимание уделяется аспектам XML Schema, непосредственно применимым к XSLT. Схемосовместимый процессор XSLT может использовать схемы для проверки документов и элементов. Если вы работаете с типами, объявленными в XML Schema, схемосовместимый процессор может использовать эти типы данных точно так же, как `xs:string`, `xs:date` или любой другой базовый тип данных.

Объявление элементов и атрибутов

Пожалуй, самой стандартной задачей в XML Schema является объявление элементов и атрибутов. Мы начнем с пустого элемента, а затем перейдем к более сложным вещам.

Создание пустого элемента

Пустые элементы создаются следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- empty1.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="empty1">
    <xs:complexType/>
```

```
  </xs:element>

</xs:schema>
```

Вот и все! Пустой элемент <xs:complexType> означает, что элемент не имеет атрибутов и содержимого. Пользы от такого элемента немного, но это другой вопрос. Документ XML, в котором используется эта схема, выглядит так:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- empty1.xml -->
<empty1
  xmlns="http://www.oreilly.com/xslt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oreilly.com/xslt empty1.xsd"/>
```

Ранее было сказано, что пустой элемент не может иметь атрибутов, однако здесь мы вроде бы видим два или три атрибута. Первые два (xmlns и xmlns:xsi) на самом деле являются объявлениями пространств имен, а не атрибутами. Атрибут xsi:schemaLocation (*действительно* атрибут) связывает схему с документом. В схеме атрибут targetNamespace определяет пространство имен, используемое схемой; оно также является пространством имен по умолчанию (xmlns="http://www.oreilly.com/xslt") в схеме и документе XML. Чтобы связать все воедино, атрибут xsi:schemaLocation в документе XML сообщает парсеру XML, где искать схему, определяющую структуру действительного документа.

Существуют и другие способы использования пространств имен, но здесь они подробно не рассматриваются. Например, документ XML может выглядеть так:

```
<ora:empty1
  xmlns:ora="http://www.oreilly.com/xslt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oreilly.com/xslt empty1.xsd"/>
```

Мы берем пространство имен по умолчанию и объявляем префикс, соответствующий пространству имен в документе XML Schema. Так как документ XML не имеет пространства имен по умолчанию, нам приходится уточнять все элементы префиксом, соответствующим целевому пространству имен схемы. Именно по этой причине элемент <empty1> превратился в <ora:empty1>.

Создание пустого элемента с атрибутами

Следующий шаг – создание атрибутов в пустом элементе. Для этого мы объявляем то, что в XML Schema называется *составным типом*:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- empty2.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="empty2">
    <xs:complexType>
      <xs:attribute name="color"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Документ XML стал немного интереснее:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- empty2.xml -->
<empty2
  xmlns="http://www.oreilly.com/xslt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oreilly.com/xslt empty2.xsd"
  color="blue"/>
```

Здесь просто объявляется атрибут с именем. Также можно было объявить тип данных атрибута; вскоре мы сделаем и это.

Создание элемента с текстом

Действительно полезные элементы обычно имеют содержимое. Начнем с создания элемента с типом `xs:string`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- content1.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="content1" type="xs:string"/>

</xs:schema>
```

Обратите внимание: на этот раз `<xs:compexType>` не используется. Мы просто указали, что элемент относится к строковому типу. Действительный документ для этой схемы выглядит так:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- content1.xml -->
<content1
  xmlns="http://www.oreilly.com/xslt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oreilly.com/xslt content1.xsd">
Our element now contains some text!  It's getting more useful
all the time.
</content1>
```

Создание элемента с текстом и атрибутами

На следующем шаге будет создан элемент, обладающий как текстом, так и атрибутами. Как говорилось ранее, для создания атрибутов необходимо использовать элемент `<xs:complexType>`, поэтому нам придется создать новый тип на базе `xs:string`. Вот как это делается:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- content2.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="content2">
```

```
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="color" type="xs:string"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Новый элемент имеет составной тип, но он использует то, что в XML Schema называется *простым содержимым*. Другими словами, элемент является расширением простого типа `xs:string`. Расширение сводится к добавлению атрибута. Новая версия документа выглядит так:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- content2.xml -->
<content2
  xmlns="http://www.oreilly.com/xslt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oreilly.com/xslt content2.xsd"
  color="blue">
Our element now contains some text!  It's getting more useful
all the time.
</content2>
```

Создание элемента со смешанным содержимым

В последнем примере создается элемент со смешанным содержимым. *Смешанное содержимое* означает, что элемент может содержать текст и другие элементы. Далее приводится расширенная схема с объявлением второго элемента; первый элемент может содержать произвольную комбинацию текста и второго элемента:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- content3.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="content3">
    <xs:complexType mixed="true">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="emphasis"/>
      </xs:sequence>
      <xs:attribute name="color" type="xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="emphasis" type="xs:string"/>

</xs:schema>
```

Мы объявляем составной тип со смешанным содержимым, элемент и атрибут. Действительный документ для новой схемы:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- content3.xml -->
```

```
<content3
  xmlns="http://www.oreilly.com/xslt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oreilly.com/xslt content3.xsd"
  color="blue">
Our element <emphasis>now</emphasis> contains some text!
It's getting <emphasis>more useful</emphasis> all the time.
</content3>
```

И последняя деталь: в схеме элемент `<emphasis>` объявляется внутри составного типа, а ссылка на него присутствует в объявлении элемента `<content3>`. Если бы элемент `<emphasis>` объявлялся следующим образом:

```
<xs:complexType mixed="true">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="emphasis" type="xs:string"/>
  ...
```

мы бы получили те же результаты, но не смогли бы использовать элемент `<emphasis>` в другом месте. Во всех дальнейших примерах мы будем объявлять элементы глобально и ссылаться на них там, где потребуется. (*Глобально объявленным элементом* называется элемент `<xs:element>`, родителем которого является элемент `<xs:schema>`.)

Определение типов данных

Некоторые элементы и атрибуты, создававшиеся нами ранее, использовали встроенные типы данных XML Schema. Механизм определения собственных типов значительно расширяет возможности языка. Пользовательские типы данных имеют самое тесное отношение к XSLT. Схемосовместимый процессор позволяет использовать собственные типы данных для проверки документа. Способы создания типов данных рассматриваются в следующем разделе.

Мы не рассматриваем базовые типы данных (`xs:string`, `xs:integer` и т. д.), используемые в XML Schema. За подробной информацией об этих типах обращайтесь к разделу «[2.0] Типы данных XPath 2.0» в приложении B.

Анонимные типы

Во всех предшествующих примерах применялись анонимные типы. Иначе говоря, мы использовали элементы `<xs:simpleType>` или `<xs:complexType>`, но не присваивали типам имена. Для простых схем это вполне нормально, поскольку типы данных не предполагается использовать за пределами того элемента, в котором они определяются. В дальнейшем мы будем присваивать типам данных имена, чтобы их можно было использовать заново. Следующая таблица стилей демонстрирует различия между анонимным и именованным типами:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- content4.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
  <xs:element name="content4a">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="color" type="xs:string"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="content4b-type">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="color" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:element name="content4b" type="content4b-type"/>

</xs:schema>
```

Первый элемент схемы – <content4a> – использует анонимный тип. Второй элемент имеет точно такую же структуру, но использует именованный тип данных, созданный нами. Различие заключается в том, что именованный тип данных можно использовать где угодно, а анонимный тип существует только внутри объявления элемента <content4a>.

Группы

В XML Schema определяются группы трех видов: группы <xs:sequence>, группы <xs:choice> и группы <xs:all>. Обычно эти группы находятся в объявлении типа или элемента, но элемент <xs:group> может также использоваться для создания автономных групп и ссылки на них по мере надобности.

Наиболее гибкой разновидностью являются группы выбора. *Группа выбора* (<xs:choice>) содержит список элементов, только один из которых может присутствовать в действительном документе XML. На первый взгляд такое ограничение кажется слишком жестким, но группы выбора могут встречаться нуль или более раз, что позволяет применять их в смешанной модели содержимого. Например, следующая группа выбора сообщает, что элемент <p> может содержать текст и произвольную комбинацию элементов <a>, <b>, <br>, <code>, <i> или <img>:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- paragraph.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="a">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="href" type="xs:string"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
```

```
  </xs:element>

  <xs:element name="b">
    <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="a"/>
        <xs:element ref="br"/>
        <xs:element ref="code"/>
        <xs:element ref="i"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:element name="br"/>

  <xs:element name="code">
    <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="a"/>
        <xs:element ref="b"/>
        <xs:element ref="br"/>
        <xs:element ref="i"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:element name="i">
    <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="a"/>
        <xs:element ref="b"/>
        <xs:element ref="br"/>
        <xs:element ref="code"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:element name="img">
    <xs:complexType>
      <xs:attribute name="href" type="xs:string"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="p">
    <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="a"/>
        <xs:element ref="b"/>
        <xs:element ref="br"/>
        <xs:element ref="code"/>
        <xs:element ref="i"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Все элементы со смешанным содержимым также могут включать произвольное количество элементов из `<xs:choice>`. Схема предоставляет значительную гибкость; на-

пример, элемент <p> может содержать текст с элементом <code>, который в свою очередь содержит элемент <b>, а последний в свою очередь содержит элемент <i>.

Вторую разновидность групп составляют группы последовательности. Группы последовательности определяются элементом <xs:sequence> и содержат список элементов, которые должны присутствовать в документе в последовательности их перечисления. Атрибут minOccurs="0" указывает, что некоторые элементы могут быть необязательными. Следующая схема определяет элементы, которые должны находиться в элементе <person>:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- person.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name"/>
        <xs:element ref="age"/>
        <xs:element ref="birthday"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="name" type="xs:string"/>
  <xs:element name="age" type="xs:positiveInteger"/>
  <xs:element name="birthday" type="xs:date"/>

</xs:schema>
```

Эта схема означает, что каждый элемент <person> должен содержать элемент <name>, элемент <age> и элемент <birthday>. Элементы должны следовать в указанном порядке, причем все они являются обязательными.

К последней разновидности относятся группы <xs:all>. Группа <xs:all> содержит список элементов. Все элементы из списка должны присутствовать один раз, но порядок их следования может быть любым. При необходимости можно пометить некоторые элементы как необязательные при помощи атрибута minOccurs="0". Атрибуты minOccurs и maxOccurs могут принимать только два значения, 0 и 1. Пример схемы с группой <xs:all>:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- person-all.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="person">
    <xs:complexType>
      <xs:all>
        <xs:element ref="name"/>
        <xs:element ref="age"/>
        <xs:element ref="birthday"/>
      </xs:all>
    </xs:complexType>
```

```
  </xs:element>

  <xs:element name="name" type="xs:string"/>
  <xs:element name="birthday" type="xs:date" nillable="true"/>
  <xs:element name="age" type="xs:positiveInteger"/>

</xs:schema>
```

Три элемента могут следовать в произвольном порядке, но присутствовать должны все три. Пример документа XML, действительного согласно этой схеме:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- person-all.xml -->
<person
  xmlns="http://www.oreilly.com/xslt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oreilly.com/xslt person-all.xsd">
  <age>42</age>
  <birthday>1965-06-19</birthday>
  <name>Doug Tidwell</name>
</person>
```

Для групп `<xs:all>` устанавливается одно важное ограничение. Элементы, определяемые или упоминаемые в группах, могут быть только отдельными элементами. Для включения нескольких элементов в группу нельзя воспользоваться другой группой – элементы необходимо включать по отдельности.

И последнее замечание, прежде чем мы расстанемся с темой групп: XML Schema определяет элемент `<xs:group>`. Он может использоваться для определения группы элементов или обращения к группе элементов, определяемой в другом месте. Пример:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- group.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="person">
    <xs:complexType>
      <xs:group ref="person-elements"/>
    </xs:complexType>
  </xs:element>

  <xs:group name="person-elements">
    <xs:sequence>
      <xs:element ref="name"/>
      <xs:element ref="birthday"/>
      <xs:element ref="age"/>
    </xs:sequence>
  </xs:group>

  <xs:element name="name" type="xs:string"/>
  <xs:element name="birthday" type="xs:date"/>
  <xs:element name="age" type="xs:positiveInteger"/>

</xs:schema>
```

Мы определяем именованную группу и ссылаемся на нее в определении элемента `<person>`.

Создание новых типов данных посредством ограничения

Одним из стандартных способов создания новых типов данных является ограничение значений другого типа. Мы рассмотрим три варианта: диапазоны, перечисления и регулярные выражения.

В первом примере диапазон используется для определения типа данных с именем age, базирующегося на xs:integer. Мы хотим, чтобы тип данных age («возраст») принимал значения от 0 до 130. (Как видите, мы оптимистично смотрим на проблему долголетия.) Диапазон допустимых значений определяется следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- person1.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name"/>
        <xs:element ref="age"/>
      </xs:sequence>
      <xs:attribute name="eyeColor" type="xs:string"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="name" type="xs:string"/>
  <xs:simpleType name="age-type">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="130"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="age" type="age-type"/>

</xs:schema>
```

Мы определяем новый тип данных age-type и используем его в качестве типа данных элемента <age>. Документ XML, действительный согласно этой схеме:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- person1.xml -->
<person
  xmlns="http://www.oreilly.com/xslt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oreilly.com/xslt person1.xsd"
  eyeColor="brown">
  <name>Doug Tidwell</name>
  <age>42</age>
</person>
```

Если значение <age> выходит за пределы определяемого диапазона, документ не пройдет проверку на действительность.

В продолжение разработки нашей схемы мы ограничим значение атрибута `eyeColor`. Для простоты атрибут будет ограничен четырьмя возможными значениями. Задача решается при помощи перечисления:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- person2.xsd -->
...
      <xs:attribute name="eyeColor" type="eyeColor-type"/>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name="eyeColor-type">
    <xs:restriction base="xs:string">
      <xs:enumeration value="blue"/>
      <xs:enumeration value="brown"/>
      <xs:enumeration value="gray"/>
      <xs:enumeration value="green"/>
    </xs:restriction>
  </xs:simpleType>
  ...
```

Если значение атрибута `eyeColor` отлично от четырех определенных значений, документ не проходит проверку.

В последнем варианте ограничений используется регулярное выражение. Оно определяет, как должны выглядеть действительные данные; если регулярное выражение не совпадает со значением заданного типа, документ не проходит проверку. Пример определения типа данных с регулярным выражением:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- person3.xsd -->
...
  <xs:element name="driversLicense" type="driversLicense-type"/>
...
  <xs:simpleType name="driversLicense-type">
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z]{2}-[0-9]{4}-[0-9]{6}"/>
    </xs:restriction>
  </xs:simpleType>
  ...
```

Документ, действительный согласно этой схеме:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- person3.xml -->
<person
  xmlns="http://www.oreilly.com/xslt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oreilly.com/xslt person3.xsd"
  eyeColor="brown">
  <name>Doug Tidwell</name>
  <age>42</age>
  <driversLicense>NC-3821-388297</driversLicense>
</person>
```

(Обратите внимание: элемент, использующий тип данных, определяется ранее самого типа данных. Порядок определения в схеме неважен.)

Так как наши типы данных определяются посредством ограничения, проверка завершается неудачей, если <age> выходит за границы заданного диапазона, *или* значение eyeColor не входит в набор из четырех цветов, *или* <driversLicense> не соответствует регулярному выражению. Так как все типы данных являются именованными, мы можем повторно использовать их по мере надобности.

Создание новых типов данных посредством расширения

Другой способ создания новых типов данных основан на расширении. Самым распространенным примером является добавление атрибута в простой тип. В спецификации XML Schema сказано, что простые типы (xs:string, xs:integer и т. д.) не могут обладать атрибутами. Мы можем создать составной тип, у которого элемент относится к типу xs:integer, но при этом обладает атрибутом. Пример схемы:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- extension.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:simpleType name="currency-type">
    <xs:restriction base="xs:string">
      <xs:enumeration value="USD"/>
      <xs:enumeration value="GBP"/>
      <xs:enumeration value="CNY"/>
      <xs:enumeration value="EUR"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="price-type">
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
        <xs:attribute name="currency" type="currency-type"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:element name="price" type="price-type"/>

</xs:schema>
```

Содержимое элемента должно относиться к типу xs:decimal, а сам элемент должен обладать атрибутом currency. Документ, действительный согласно этой схеме:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- extension.xml -->
<price
  xmlns="http://www.oreilly.com/xslt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oreilly.com/xslt extension.xsd"
  currency="USD">
438.92
</price>
```

Если изменить содержимое элемента на akd482.58, проверка завершится неудачей.

Преобразования типов данных

Атомарное значение одного атомарного типа данных можно преобразовать к другому типу. С составными типами такое преобразование невозможно, но для атомарных типов оно работает. Короткий пример, демонстрирующий преобразование типа:

```
<?xml version="1.0"?>
<!-- typecasting.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:age="http://www.oreilly.com/xslt"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:import-schema namespace="http://www.oreilly.com/xslt">
    <xs:schema
      targetNamespace="http://www.oreilly.com/xslt"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:simpleType name="age-type">
        <xs:restriction base="xs:integer">
          <xs:minInclusive value="0"/>
          <xs:maxInclusive value="130"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:schema>
  </xsl:import-schema>

  <xsl:template match="/">
    <xsl:variable name="age" as="age:age-type"
      select="age:age-type(42)"/>
    <xsl:variable name="age-int" as="xs:integer"
      select="$age cast as xs:integer"/>
    <xsl:variable name="float-age" as="age:age-type"
      select="xs:float(42.0) cast as age:age-type"/>
    <xsl:value-of select="$age, $age-int, $float-age"
      separator=", "/>
  </xsl:template>
</xsl:stylesheet>
```

Три переменные в конце таблицы стилей относятся к типам `age:age-type`, `xs:integer` и `xs:float`. Так как все типы являются числовыми, значение одного типа можно преобразовать к другому типу. При этом должны соблюдаться ограничения для каждого типа данных; команда `xs:float(148.3) cast as age:age-type` приведет к ошибке во время выполнения. Преобразование `xs:float(48.3)` к типу `age:age-type` создает новое значение типа `age:age-type` со значением `48`.

Чтобы «преобразовать» значение к составному типу или из него, необходимо построить составной тип самостоятельно. Другими словами, чтобы создать и проверить на действительность составной тип, необходимо использовать `<xsl:element>` с соответствующим заполнением элементов и атрибутов. За дополнительной информацией обращайтесь ко второму примеру элемента `<xsl:element>` в приложении A.

Создание списковых типов

Другую разновидность типов данных, которые мы можем создавать, составляют списковые типы. Списковый тип ссылается на некоторый тип данных; новый списковый

тип содержит значения этого типа, разделенные пробелами. Давайте включим в нашу схему новый тип данных с именем state-abbr, а затем создадим на его основе списковый тип. Схема:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- list.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name"/>
        <xs:element ref="birthday"/>
        <xs:element ref="age"/>
        <xs:element ref="priorAddresses"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name="state-abbr-type">
    <xs:restriction base="xs:string">
      <xs:length value="2"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="state-abbr-list-type">
    <xs:list itemType="state-abbr-type"/>
  </xs:simpleType>

  <xs:element name="name" type="xs:string"/>
  <xs:element name="birthday" type="xs:date"/>
  <xs:element name="age" type="xs:positiveInteger"/>
  <xs:element name="priorAddresses">
    <xs:complexType>
      <xs:attribute name="states" type="state-abbr-list-type"/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Теперь в документе появился атрибут с именем states, содержимое которого представляет собой список из одного или нескольких сокращений штатов. Действительный документ:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- list.xml -->
<person
  xmlns="http://www.oreilly.com/xslt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oreilly.com/xslt list.xsd">
  <name>Doug Tidwell</name>
  <birthday>1965-06-19</birthday>
  <age>42</age>
  <priorAddresses states="GA TN NC"/>
</person>
```

Создание объединений

Тип объединение берет за основу два или более типов данных. Действительным содержимым объединения является значение одного из этих типов. Мы создадим другую схему, в которой определяется список почтовых индексов, а затем создадим объединение, которое разрешает атрибуту `states` содержать список сокращений штатов или список почтовых индексов. Фрагмент схемы:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- union.xsd -->
...
  <xs:simpleType name="zipcode">
    <xs:restriction base="xs:integer">
      <xs:pattern value="[0-9]{5}"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="zipcode-list-type">
    <xs:list itemType="zipcode"/>
  </xs:simpleType>

  <xs:simpleType name="state-abbr-type">
    <xs:restriction base="xs:string">
      <xs:length value="2"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="state-abbr-list-type">
    <xs:list itemType="state-abbr-type"/>
  </xs:simpleType>

  <xs:simpleType name="address-union-type">
    <xs:union memberTypes="state-abbr-list-type zipcode-list-type"/>
  </xs:simpleType>
  ...
  <xs:element name="priorAddresses">
    <xs:complexType>
      <xs:attribute name="states" type="address-union-type"/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Документ, действительный согласно этой схеме:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- union.xml -->
<person
  xmlns="http://www.oreilly.com/xslt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oreilly.com/xslt union.xsd">
  <name>Doug Tidwell</name>
  <birthday>1965-06-19</birthday>
  <age>42</age>
  <priorAddresses states="27516 37174 30606"/>
</person>
```

Атрибут states, содержащий либо список почтовых индексов, либо список сокращений, допустим. *Нельзя* лишь объединять разнородные данные. Следующий пример недопустим:

```
<!-- Недействительное значение! -->
<priorAddresses states="27516 37174 30606 NC TN GA"/>
```

Группы подстановки

Группой подстановки называется набор элементов, которые могут подставляться на место друг друга. Мы включим в схему элемент <address>, а затем определим два альтернативных типа адресов. Схема:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- substitute.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name"/>
        <xs:element ref="birthday"/>
        <xs:element ref="age"/>
        <xs:element ref="address"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="name" type="xs:string"/>
  <xs:element name="birthday" type="xs:date"/>
  <xs:element name="age" type="xs:positiveInteger"/>

  <xs:element name="address" type="xs:string"/>
  <xs:element name="businessAddress" type="xs:string"
    substitutionGroup="address"/>
  <xs:element name="residentialAddress" type="xs:string"
    substitutionGroup="address"/>

</xs:schema>
```

Схема определяет два дополнительных элемента <businessAddress> и <residentialAddress>, которые могут подставляться на место <address>. Обратите внимание: два дополнительных элемента используют атрибут substitutionGroup для ссылки на заменяемый элемент. Кроме того, все эти элементы относятся к одному типу. Документ, действительный согласно этой схеме:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- substitute.xml -->
<person
  xmlns="http://www.oreilly.com/xslt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oreilly.com/xslt substitute.xsd">
  <name>Doug Tidwell</name>
  <birthday>1965-06-19</birthday>
```

```
  <age>42</age>
  <address>4013 Corporate Parkway</address>
</person>
```

Следующий документ тоже является действительным:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- substitute2.xml -->
<person
...
  <businessAddress>4013 Corporate Parkway</businessAddress>
</person>
```

И этот документ тоже действителен:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- substitute3.xml -->
<person
...
  <residentialAddress>1234 Main Street</residentialAddress>
</person>
```

Абстрактные элементы и типы данных

Последняя тема этого раздела – *абстрактные элементы и типы данных*. Абстрактный элемент или тип данных напоминает абстрактный класс в объектно-ориентированных языках программирования: абстрактный класс определяется в программе, но для его реального использования необходимо создать подкласс. В контексте XML Schema новые элементы и типы данных создаются на базе абстрактных прототипов.

В первом примере мы определим абстрактный элемент и его свойства, а затем используем группу подстановки, как делалось ранее. Единственное изменение в схеме заключается в том, что в определение элемента `<address>` добавляется атрибут `abstract="true"`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- abstract1.xsd -->
<xs:schema
...
  <xs:element name="address" type="xs:string" abstract="true"/>

  <xs:element name="businessAddress" type="xs:string"
    substitutionGroup="address"/>
  <xs:element name="residentialAddress" type="xs:string"
    substitutionGroup="address"/>
...
```

Включая абстрактный элемент, мы фактически требуем, чтобы документ содержал либо `<businessAddress>`, либо `<residentialAddress>`. Пример действительного документа:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- abstract1.xml -->
<person
  xmlns="http://www.oreilly.com/xslt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oreilly.com/xslt abstract1.xsd">
  <name>Doug Tidwell</name>
  <birthday>1965-06-19</birthday>
```

```
    <age>42</age>
    <businessAddress>4013 Corporate Parkway</businessAddress>
</person>
```

Если документ содержит элемент `<address>` **вместо** `<businessAddress>`**, он не проходит проверку на действительность.**

Перейдем к абстрактным типам данных. В следующем примере определяются абстрактный тип данных `state-or-province-type` **и элемент этого типа с именем** `<state-or-province>`**. Затем мы определим два неабстрактных расширения этого типа с именами** `province-type` **и** `state-type`**. Элементы** `<xsl:state-or-province>` **в следующем примере используют атрибут** `xsi:type` **для объявления элементов неабстрактного типа:**

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- abstract2.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name"/>
        <xs:element ref="birthday"/>
        <xs:element ref="age"/>
        <xs:element ref="state-or-province"
          minOccurs="2" maxOccurs="2"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="name" type="xs:string"/>
  <xs:element name="birthday" type="xs:date"/>
  <xs:element name="age" type="xs:positiveInteger"/>

  <xs:element name="address" type="xs:string"/>

  <xs:complexType name="state-or-province-type" abstract="true"/>

  <xs:complexType name="state-type">
    <xs:complexContent>
      <xs:extension base="state-or-province-type">
        <xs:attribute name="state-abbr">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="NC"/>
              <xs:enumeration value="TN"/>
              <xs:enumeration value="GA"/>
              <!-- Другие штаты не указаны -->
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="province-type">
    <xs:complexContent>
```

```
      <xs:extension base="state-or-province-type">
        <xs:attribute name="province-abbr">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="NS"/>
              <xs:enumeration value="BC"/>
              <xs:enumeration value="PEI"/>
              <!-- Другие провинции не указаны -->
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="state-or-province" type="state-or-province-type"/>

</xs:schema>
```

Определяемые типы данных расширяют абстрактный тип добавлением атрибута. Каждый атрибут ограничивается посредством перечисления. Тип данных `state-type` содержит атрибут `state-abbr`, который принимает только значения из фиксированного набора; то же относится к типу данных `province-type` и к его атрибуту `province-abbr`.

Пример действительного документа XML:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- abstract2.xml -->
<person
  xmlns="http://www.oreilly.com/xslt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oreilly.com/xslt abstract2.xsd">
  <name>Doug Tidwell</name>
  <birthday>1965-06-19</birthday>
  <age>42</age>
  <state-or-province xsi:type="province-type" province-abbr="NS"/>
  <state-or-province xsi:type="state-type" state-abbr="NC"/>
</person>
```

(Для сохранения наглядности мы изменили атрибут `minOccurs` элемента `<state-or-province>`, чтобы в нем могли использоваться оба неабстрактных типа.) Поскольку элемент `<state-or-province>` объявляется с абстрактным типом данных, нам придется использовать атрибут `xsi:type`, чтобы объявить неабстрактный тип данных для элемента.

Использование схемы в таблице стилей

Итак, от рассмотрения общих принципов работы схем мы переходим к их использованию в документах XSLT. Элемент `<xsl:import-schema>` позволяет напрямую импортировать XML Schema. Помимо элемента `<xsl:import-schema>`, документ XML может ссылаться на схему при помощи атрибутов `xsi:noNamespaceSchemaLocation` или `xsi:schemaLocation`. Если таблица стилей имеет правильную конфигурацию, процессор XSLT сможет использовать указанную схему для проверки действительности данных.

Импортирование схем элементом <xsl:import-schema>

Элемент <xsl:import-schema> импортирует документ XML Schema. Импортирование и обработка схемы выполняются до обработки каких-либо входных документов. Это позволяет определять типы данных и правила проверки до обработки и преобразования исходного документа процессором XSLT. *Данный элемент поддерживается только схемосовместимыми процессорами XSLT 2.0.*

Элемент имеет два необязательных параметра: namespace, определяющий URI-адрес пространства имен схемы, и schema-location, содержащий URI-адрес самого файла схемы. Элемент <xsl:import-schema> использует schema-location для импортирования файла или содержит определение XML Schema непосредственно в таблице стилей. Мы рассмотрим пару примеров; за более подробным описанием обращайтесь к описанию элемента *[2.0 – Schema]* <xsl:import-schema>.

Простейший способ импортирования схемы в таблицу стилей заключается в использовании URI-адреса:

```
<?xml version="1.0"?>
<!-- import-schema.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:po="http://www.oreilly.com/xslt">

  <xsl:import-schema namespace="http://www.oreilly.com/xslt"
    schema-location="po.xsd" />

  <xsl:output method="text"/>

  <xsl:template match="schema-element(po:purchase-order)">
    <xsl:text>&#xA;This is a test of the &lt;xsl:import-</xsl:text>
    <xsl:text>schema&gt; element.&#xA;&#xA;</xsl:text>
    <xsl:text>Here are all the items in this purchase </xsl:text>
    <xsl:text>order:&#xA;</xsl:text>
    <xsl:for-each select="po:items/po:item">
      <xsl:text>  * </xsl:text>
      <xsl:value-of select="po:partname"/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

  <xsl:template match="*">
    <xsl:message terminate="yes">
      <xsl:text>This is not a valid purchase order!</xsl:text>
    </xsl:message>
  </xsl:template>

</xsl:stylesheet>
```

Обратите внимание, что импортируемая схема связывается с URI-адресом пространства имен, который в свою очередь связывается с префиксом po. Шаблон XSLT использует тест узла schema-element(po:purchase-order) для поиска всех элементов с именем <purchase-order> и URI-адресом пространства имен http://www.oreilly.com/xslt. С этой отправной точки таблица стилей находит все элементы <item> в заказе. Результат выглядит так:

```
This is a test of the <xsl:import-schema> element.

Here are all the items in this purchase order:
```

```
    * Turnip Twaddler
    * Clam Teaser
```

Второй шаблон в таблице стилей не совпадает ни с чем, *кроме* действительного заказа. Если корневой элемент не имеет правильного имени и URI-адреса пространства имен, второй шаблон останавливает процессор.

Элемент `<xsl:import-schema>` может также использоваться со схемой, встроенной в таблицу стилей XSLT:

```
<?xml version="1.0"?>
<!-- import-schema2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns="http://www.oreilly.com/xslt"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:po="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:import-schema namespace="http://www.oreilly.com/xslt">
    <xs:schema
      targetNamespace="http://www.oreilly.com/xslt"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">

      <xs:element name="purchase-order">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="date"
              minOccurs="1" maxOccurs="1"/>
            <xs:element ref="customer"
              minOccurs="1" maxOccurs="1"/>
            <xs:element ref="items"
              minOccurs="1" maxOccurs="1"/>
          </xs:sequence>
          <xs:attribute name="id" type="xs:string"/>
        </xs:complexType>
      </xs:element>

      <xs:element name="date">
        <xs:complexType>
          <xs:attribute name="year" type="xs:integer"/>
          <xs:attribute name="month" type="xs:integer"/>
          <xs:attribute name="day" type="xs:integer"/>
        </xs:complexType>
      </xs:element>
...
    </xs:schema>
  </xsl:import-schema>
...
  <xsl:template match="schema-element(po:purchase-order)">
    ...
</xsl:stylesheet>
```

Если элемент `<xsl:import-schema>` имеет как атрибут `select`, так и содержимое, происходит фатальная ошибка.

Использование схем XML без пространств имен

Чтобы использовать документ XML Schema без элемента `<xsl:import-schema>`, необходимо связать воедино XML, схему и таблицу стилей. Для начала мы сделаем это без использования пространств имен, начиная с атрибута `xsi:noNamespaceSchemaLocation` в документе XML:

```
<?xml version="1.0"?>
<!-- parts-list-schema-no-ns.xml -->
<parts-list
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="parts-list-no-ns.xsd">
  <component component-id="C28392-33-TT">
    <name>Turnip Twaddler</name>
    <partref refid="P81952-26-PK"/>
    <partref refid="P86679-52-SP"/>
    ...
```

Схема не задает пространство имен по умолчанию:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- parts-list-no-ns.xsd -->
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="parts-list">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="component" minOccurs="1" maxOccurs="unbounded"/>
        ...
```

Единственный префикс, который используется в этом фрагменте – префикс самого документа XML Schema. Файл XML и схема синхронизированы, поэтому оставшаяся часть таблицы стилей весьма тривиальна:

```
<?xml version="1.0"?>
<!-- id-schema-no-ns.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>
  ...
    <xsl:for-each select="/parts-list/part">
      <xsl:text>&#xA;  </xsl:text>
      <xsl:value-of select="name"/>
      <xsl:text> (part #</xsl:text>
      <xsl:value-of select="@part-id"/>
      <xsl:text>) is used in these products:&#xA;    </xsl:text>
      <xsl:for-each
        select="/parts-list/component
                [partref/@refid=current()/@part-id]">
        <xsl:value-of select="name"/>
        ...
```

Так как пространства имен не используются, нам не приходится ничего уточнять в выражениях XPath. Использование схемы с пространствами имен рассматривается в следующем подразделе.

Использование схем XML с пространствами имен

При использовании документа XML Schema с пространством имен необходимо обеспечить синхронизацию всех трех файлов (*.xml*, *.xsd* и *.xsl*). Как нетрудно догадаться, задача немного усложняется. Прежде всего, файл XML должен иметь пространство имен по умолчанию:

```
<?xml version="1.0"?>
<!-- parts-list-schema-ns.xml -->
<parts-list xmlns="http://www.oreilly.com/xslt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oreilly.com/xslt
                      parts-list-schema-ns.xsd">
  <component component-id="C28392-33-TT">
    <name>Turnip Twaddler</name>
    <partref refid="P81952-26-PK"/>
    ...
```

Обратите внимание на определение пространства имен по умолчанию (`xmlns=`) со значением `http://www.oreilly.com/xslt`. Атрибут `xsi:schemaLocation` состоит из двух частей, разделенных пропусками: URI-адреса пространства имен и URI-адреса самой схемы. Файл схемы использует то же пространство имен по умолчанию:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- parts-list-schema-ns.xsd -->
<xs:schema
  xmlns="http://www.oreilly.com/xslt"
  targetNamespace="http://www.oreilly.com/xslt"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="parts-list">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="component" minOccurs="1" maxOccurs="unbounded"/>
        <xs:element ref="part" minOccurs="1" maxOccurs="unbounded"/>
        ...
```

Также мы используем атрибут XML Schema `targetNamespace`; его значение совпадает с пространством имен по умолчанию. Наконец, пространство имен должно использоваться и в файле XSLT. Мы определяем `http://www.oreilly.com/xslt` как пространство имен по умолчанию и задаем для него префикс:

```
<?xml version="1.0"?>
<!-- id-schema-ns.xsl -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.oreilly.com/xslt"
  xmlns:pl="http://www.oreilly.com/xslt">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Here is a test of the id() </xsl:text>
    <xsl:text>function in reverse:&#xA;</xsl:text>
    <xsl:for-each select="/pl:parts-list/pl:part">
      <xsl:text>&#xA;  </xsl:text>
      <xsl:value-of select="pl:name"/>
```

```
    <xsl:text> (part #</xsl:text>
    <xsl:value-of select="@part-id"/>
    <xsl:text>) is used in these products:&#xA;    </xsl:text>
    <xsl:for-each
      select="/pl:parts-list/pl:component
              [pl:partref/@refid=current()/@part-id]">
      <xsl:value-of select="pl:name"/>
```

Так как в схеме определяются пространства имен, мы должны создать для пространства имен префикс и использовать его при всех ссылках на элементы документа XML. Если изменить таблицу стилей так, чтобы вместо элементов `<pl:parts-list>` она искала `<parts-list>`, таблица стилей работать не будет. Использование схемы с пространствами имен создает дополнительные трудности, но у вас не всегда есть выбор.

Для решения описанной выше проблемы в XSLT 2.0 был добавлен атрибут `xpath-default-namespace`. В предыдущей таблице стилей мы определяли пространство имен и префикс, а затем использовали префикс во всех командах XPath. С атрибутом `xpath-default-namespace` это делать необязательно:

```
<?xml version="1.0"?>
<!-- id-schema-ns2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.oreilly.com/xslt"
  xpath-default-namespace="http://www.oreilly.com/xslt">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Here is a test of the id() </xsl:text>
    <xsl:text>function in reverse:&#xA;</xsl:text>

    <xsl:for-each select="/parts-list/part">
      <xsl:text>&#xA;  </xsl:text>
      <xsl:value-of select="name"/>
      <xsl:text> (part #</xsl:text>
      <xsl:value-of select="@part-id"/>
      <xsl:text>) is used in these products:&#xA;    </xsl:text>
      <xsl:for-each
        select="/parts-list/component
                [partref/@refid=current()/@part-id]">
        <xsl:value-of select="name"/>
        <xsl:if test="position() != last()">
          <xsl:text>&#xA;    </xsl:text>
        </xsl:if>
      </xsl:for-each>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Включение атрибута `xpath-default-namespace` в таблицу стилей значительно упрощает выражения XPath.

Учтите, что этот атрибут может включаться в любые элементы пространства имен XSLT. Например, можно удалить атрибут `xpath-default-namespace` из элемента `<xsl:stylesheet>` и записать таблицу стилей следующим образом:

```
<?xml version="1.0"?>
<!-- id-schema-ns3.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.oreilly.com/xslt">
...
    <xsl:for-each select="/parts-list/part"
      xpath-default-namespace="http://www.oreilly.com/xslt">
      <xsl:text>&#xA;  </xsl:text>
      <xsl:value-of select="name"/>
...
</xsl:stylesheet>
```

Эта таблица стилей выдает тот же результат, что и предыдущая. Если вы используете данные из разных пространств имен (например, при чтении входных данных из нескольких документов XML), переопределение пространства имен XPath по умолчанию значительно упрощает выражения XPath. И все же в большинстве случаев этот атрибут используется в элементе `<xsl:stylesheet>`.

E

[2.0] Регулярные выражения

Язык регулярных выражений, применяемый в XSLT 2.0, XPath 2.0 и XQuery 1.0, представляет собой надмножество языка регулярных выражений XML Schema. Регулярные выражения используются в элементе XSLT 2.0 *[2.0]* `<xsl:analyze-string>` и функциях XPath 2.0 и XQuery 1.0 *[2.0]* `matches()`, *[2.0]* `replace()` и *[2.0]* `tokenize()`.

В этом приложении описываются синтаксис и возможности регулярных выражений XSLT 2.0, XPath 2.0 и XQuery 1.0. Более подробная информация о регулярных выражениях приводится в спецификации XQuery 1.0 and XPath 2.0 Functions and Operators, раздел 7.6 «String Functions that Use Pattern Matching».

[XPath] Спецификация Functions and Operators расширяет регулярные выражения XML Schema в нескольких аспектах. Возможности, уникальные для языка регулярных выражений XPath 2.0 и XQuery 1.0, обозначаются префиксом *[XPath]*.

Простые выражения

Простейшее регулярное выражение представляет собой обычную строку текста. Регулярное выражение `abc` совпадает со строкой `abc`. Фактически мы просто ищем последовательность символов в некоторых данных. В данном случае того же результата можно добиться вызовом функции `contains('abc', 'abc')`.

Однако возможности регулярных выражений выходят далеко за рамки поиска по литеральным строкам; они позволяют искать данные по образцу. Например, можно определять символьные классы; регулярное выражение `[abc]d` совпадает с двумя символами: одним из символов `a`, `b` или `c`, за которым следует символ `d`.

Символьный класс можно инвертировать; результат означает любой символ, *не входящий* в символьный класс. Инверсия выполняется включением символа ^ в начало символьного класса. Так, регулярное выражение `[^abc]d` совпадает с двумя символами: любым символом, кроме `a`, `b` и `c`, за которым следует символ `d`.

Диапазоны обеспечивают сокращенную запись для определения символьных классов. Так, символьный класс `[0-9]` обозначает любую цифру от 0 до 9, а символьный

класс [a-zA-Z] — любую букву без диакритических знаков из западноевропейских языков. Диапазоны инвертируются точно так же, как любые другие символьные классы; так, [^0-9] обозначает любой символ, кроме цифры.

Для диапазонов поддерживается также операция вычитания. Диапазон [A-Z-[IOQ]] обозначает любую букву верхнего регистра без диакритического знака, кроме I, O или Q.

Подвыражения

Регулярные выражения часто разбиваются на подвыражения. Подвыражение заключается в круглые скобки, и применение к нему квантификаторов определяет количество возможных совпадений. Например, следующее регулярное выражение совпадает с телефонным номером в формате 999-999-9999:

```
([0-9]{3})-([0-9]{3})-([0-9]{4})
```

Регулярное выражение содержит три подвыражения, каждое из которых совпадает с группой цифр. XSLT 2.0 предоставляет функцию regex-group() для получения части проанализированной строки, совпавшей с заданным подвыражением. Пример:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- subexpressions1.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:analyze-string select="'Call me at 919-555-1212, please.'"
      regex="([0-9]{{3}})-(\p{{Nd}}{{3}})-([0-9]{{4}})">
      <xsl:matching-substring>
        <xsl:text>The matching substring is '</xsl:text>

        <!-- <xsl:value-of select="."/> здесь делает то же самое -->
        <xsl:value-of select="regex-group(0)"/>
        <xsl:text>'&#xA;</xsl:text>
        <xsl:text>The formatted string is '</xsl:text>
        <xsl:value-of
          select="('(', regex-group(1), ') ', regex-group(2),
                   '-', regex-group(3), '''')"
          separator=""/>
      </xsl:matching-substring>
    </xsl:analyze-string>
  </xsl:template>

</xsl:stylesheet>
```

(Фигурные скобки удваиваются, чтобы процессор XSLT мог отличить их от шаблона значения атрибута.) Обратите внимание: регулярное выражение содержит два дефиса, не являющихся частью какого-либо подвыражения. Функция regex-group(0) возвращает полный текст совпавшей части строки, т. е. часть строки от первого символа, с которым совпало регулярное выражение, до последнего символа совпадения. В нашем примере regex-group(0) возвращает 919-555-1212 без вхождения остальных символов строки. (Запрос контекстного объекта приводит к тому же результату.) В результат включены два дефиса, не входящие ни в одно из подвыражений. Результат применения таблицы стилей:

```
The matching substring is '919-555-1212'
The formatted string is '(919) 555-1212'
```

Если значение аргумента отрицательно или превышает количество подвыражений, regex-group() возвращает пустую строку.

Функция replace() реализует механизм замены, сходный с regex-group(). В строке замены знак доллара ($) обозначает совпадения различных подвыражений. Как и следовало ожидать, $1 обозначает совпадение первого подвыражения, $2 – совпадение второго подвыражения, а $0 обозначает всю совпавшую строку. Вот как выглядит версия предыдущего примера с использованием replace():

```
<?xml version="1.0" encoding="utf-8"?>
<!-- subexpressions2.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>The string '</xsl:text>
    <xsl:value-of
      select="replace('Call me at 919-555-1212, please',
              '([0-9]{3})-([0-9]{3})-([0-9]{4})',
              '$0')"/>
    <xsl:text>' contains a match!&#xA;</xsl:text>
    <xsl:text>The version with all the replacements: &#xA;  </xsl:text>
    <xsl:value-of
      select="replace('Call me at 919-555-1212, please',
              '([0-9]{3})-([0-9]{3})-([0-9]{4})',
              '($1) $2-$3')"/>
  </xsl:template>

</xsl:stylesheet>
```

В первом вызове функции replace() используется оператор $0. Эта строка идентична исходной строке. Функция replace() заменяет только совпавшую часть строки, и замена совпадения текстом того же совпадения ничего не изменяет. При втором вызове replace() мы форматируем телефонный номер по своему усмотрению. Код зоны ($1) заключается в круглые скобки, за ним следуют пробел, код телефонной станции ($2), дефис и последние четыре цифры ($3). Результат выглядит так:

```
The string 'Call me at 919-555-1212, please' contains a match!
The version with all the replacements:
  Call me at (919) 555-1212, please
```

Между regex-group() и replace() существует одно важное различие. Вызов regex-group() с несуществующим номером группы возвращает пустую строку. Со знаком $ процессор XSLT 2.0 интерпретирует крайние правые цифры как литеральные символы и продолжает обработку до тех пор, пока в обозначении подвыражения не останется только одна цифра. Если эта цифра больше количества подвыражений, цифра заменяется пустой строкой. Пример из спецификации: если выражение содержит пять подвыражений, $23 возвращает значение второго подвыражения, за которым следует цифра 3. Обозначение $63 с тем же регулярным выражением просто возвращает значение 3.

Квантификаторы

Квантификатор указывает, сколько раз в целевом тексте встречается некоторый компонент регулярного выражения (и встречается ли вообще). Существуют три метасимвола квантификаторов и синтаксис явного определения количества экземпляров. Например, `(a|b)?c` совпадает с текстом `ac`, `bc` и `c`, потому что вопросительный знак представляет нуль или один экземпляр подвыражения. Квантификаторы и их синтаксис:

`?` Нуль или один экземпляр.

`*` Нуль или более экземпляров.

`+` Один или более экземпляров.

`{x}` Ровно `x` экземпляров. Шаблон `(a|b){3}` совпадает со строками `aaa` и `abb`, но не со строкой `ab`.

`{x,}` Не менее `x` экземпляров. Шаблон `(a|b){3,}` совпадает со строками `aaa` и `aaaaaaaaa`, но не со строкой `aa`.

`{x,y}` Не менее `x` и не более `y` экземпляров. Шаблон `(a|b){2,5}` совпадает со строками `aa` и `aaa`, но не со строкой `aaaaaa`.

[XPath] Минимальные квантификаторы

По умолчанию регулярное выражение или подвыражение совпадает с самой длинной строкой из всех возможных. Квантификаторы `+`, `*` и `?` совпадают с максимально возможным количеством символов. Если поставить после квантификатора вопросительный знак, выражение или подвыражение будет совпадать с самой *короткой* строкой из всех возможных. Модифицированные квантификаторы:

`a+?` Один экземпляр `a`.

`a??` Один или нуль экземпляров `a` (работает только в функции `matches()`; см. далее в этом разделе).

`a*?` Нуль или один экземпляр `a` (работает только в функции `matches()`; см. далее в этом разделе).

`a{x}?` Ровно `x` экземпляров `a` (в данном случае применение минимального квантификатора ни на что не влияет).

`a{x,}?` Ровно `x` экземпляров `a` (наличие запятой несущественно – минимальный квантификатор находит совпадение минимальной длины).

`a{x,y}?` Ровно `x` экземпляров `a` (наличие второго числа `y` несущественно – минимальный квантификатор находит совпадение минимальной длины).

В следующем примере используется функция `replace()`. Каждое совпадение в исходной строке заключается в квадратные скобки. Как вы убедитесь, обычные (максимальные) и минимальные квантификаторы находят разные совпадения. Таблица стилей:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- reluctant.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>
```

```
  <xsl:template match="/">
    <xsl:text>Original string: 'Call me at 19195551212'&#xA;</xsl:text>
    <xsl:text>  replace($x, '([0-9]+)', '[$1]'):&#xA;    </xsl:text>
    <xsl:value-of
      select="replace('Call me at 19195551212',
              '([0-9]+)',
              '[$1]')"/>
    <xsl:text>&#xA;</xsl:text>
    <xsl:text>  replace($x, '(([0-9])+?)', '[$1]'):&#xA;    </xsl:text>
    <xsl:value-of
      select="replace('Call me at 19195551212',
              '(([0-9])+?)',
              '[$1]')"/>
    <xsl:text>&#xA;</xsl:text>
    <xsl:text>  replace($x, '(([0-9]){2,4}?)', '[$1]'):&#xA;    </xsl:text>
    <xsl:value-of
      select="replace('Call me at 19195551212',
              '(([0-9]){2,4}?)',
              '[$1]')"/>
  </xsl:template>

</xsl:stylesheet>
```

Результат:

```
Original string: 'Call me at 19195551212'
  replace($x, '([0-9]+)', '[$1]'):
    Call me at [19195551212]
  replace($x, '(([0-9])+?)', '[$1]'):
    Call me at [1][9][1][9][5][5][5][1][2][1][2]
  replace($x, '(([0-9]){2,4}?)', '[$1]'):
    Call me at [19][19][55][51][21]2
```

С обычным квантификатором (`([0-9]+)`) совпадение находится для всей цепочки цифр, которая заключается в квадратные скобки. Во втором примере минимальный квантификатор (`(([0-9])+?)`) находит совпадение для каждой отдельной цифры, поэтому каждая цифра заключается в квадратные скобки. Последний пример ищет минимальное совпадение длиной от двух до четырех цифр; это означает, что каждое совпадение будет иметь минимальную длину в две цифры. Обратите внимание: последняя цифра в строке вообще не считается частью совпадения.

За исключением функции `matches()`, регулярное выражение не может совпадать со строкой нулевой длины. Минимальные квантификаторы `a??`, `a*?`, `a{0}?`, `a{0,}?` и `a{0,5}?` не совпадают ни с чем. Если вы используете регулярное выражение, совпадающее со строкой нулевой длины, где угодно, кроме функции `matches()`, процессор XSLT выдаст статическую ошибку.

Режимы обработки

Четыре флага управляют обработкой регулярных выражений:

`s` Регулярное выражение обрабатывается в режиме «совпадения точки со всеми символами». При наличии этого флага метасимвол «точка» (`.`) совпадает с любым символом. При стандартной обработке (без флага `s`) оператор «точка» сов-

падает с любым символом, *кроме* символа новой строки (
). Этот флаг особенно полезен при обработке строк, содержащих внутренние символы новой строки.

В Perl и других языках программирования этот режим называется «однострочным» (single-line), отсюда и выбор буквы s.

m Регулярное выражение обрабатывается в многострочном режиме. В режиме по умолчанию метасимвол ^ совпадает с началом всей строки, а метасимвол $ совпадает с концом каждой строки. В многострочном режиме ^ совпадает с началом, а $ – с концом любой логической строки внутри «физической» строки.

i Регулярное выражение обрабатывается без учета регистра символов. Регулярное выражение "a" совпадает как с "a", так и с "A".

Учтите, что Юникод может создать массу проблем с этим флагом. Например, в спецификации XQuery 1.0 and XPath 2.0 Functions and Operators приводится пример с обозначением в Юникоде градусов по Кельвину (K), соответствующим букве "K". Комбинация regex="k" и flags="i" совпадает как со знаком градусов по Кельвину, так и с буквами "k" (k) и "K" (K).

Другие символы Юникода не преобразуются в буквы. Например, символ римской цифры I (ࡰ) внешне похож на букву I, но не преобразуется в нее.

x Перед поиском совпадения из регулярного выражения удаляются все символы пропусков (	,
,
 и). Другими словами, с флагом x регулярные выражения "John Smith" и "JohnSmith" эквивалентны. Этот флаг особенно удобен при разбиении длинного регулярного выражения на несколько строк, чтобы его было легче читать.

Флаги могут комбинироваться в любом порядке. Атрибуты flags="xis" и flags="six" работают абсолютно одинаково.

[XPath] Якори

В XML Schema регулярные выражения привязываются к границам текста; иначе говоря, предполагается, что регулярное выражение начинается от начала текста и завершается в его конце. Таким образом, выражение abc совпадает только со строкой из трех символов abc. Если до или после букв abc имеются другие символы, XML Schema игнорирует такое совпадение.

Язык регулярных выражений XPath 2.0 работает не так. При использовании любых функций или элементов, поддерживающих регулярные выражения, совпадение считается найденным, если регулярное выражение совпадает *где угодно* внутри строки. Иначе говоря, на языке регулярных выражений XPath совпадение для выражения abc будет найдено как в строке abc, так и в строке I know my abc. XPath поддерживает традиционные якорные метасимволы, используемые в других языках регулярных выражений. Метасимвол ^ совпадает с началом строки, а знак $ – с концом строки. При включении многострочного режима (-m) метасимвол ^ совпадает с началом «физической» строки и с позицией за любым внутренним символом новой строки. Аналогично в многострочном режиме знак $ совпадает с концом «физической» строки и с позицией непосредственно перед внутренним символом новой строки.

Следующий пример демонстрирует работу якорных метасимволов:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- anchors.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>matches('abcdefghij', 'cde'): </xsl:text>
    <xsl:value-of
      select="if (matches('abcdefghij', 'cde'))
              then 'It''s a match!'
              else 'No match'"/>
    <xsl:text>&#xA;matches('abcdefghij', 'cde$'): </xsl:text>
    <xsl:value-of
      select="if (matches('abcdefghij', 'cde$'))
              then 'It''s a match!'
              else 'No match'"/>
    <xsl:text>&#xA;matches('abcdefghij', 'hij$'): </xsl:text>
    <xsl:value-of
      select="if (matches('abcdefghij', 'hij$'))
              then 'It''s a match!'
              else 'No match'"/>
    <xsl:text>&#xA;matches('ab&amp;#xA;cdefghij', '^cde', 'm'): </xsl:text>
    <xsl:value-of
      select="if (matches('ab&#xA;cdefghij', '^cde', 'm'))
              then 'It''s a match!'
              else 'No match'"/>
  </xsl:template>

</xsl:stylesheet>
```

В первом примере якоря вообще не используются, поэтому `matches()` возвращает `true`. Второй пример ищет символы `cde` в конце строки; в этом случае `matches()` возвращает `false`. В третьем примере символы `hij` находятся в конце строки, поэтому совпадение имеется. В последнем примере в строку вставляется символ новой строки (`
`), а поиск ведется в многострочном режиме. Поиск `cde` в начале строки завершается успехом. Результат применения таблицы стилей:

```
matches('abcdefghij', 'cde'): It's a match!
matches('abcdefghij', 'cde$'): No match
matches('abcdefghij', 'hij$'): It's a match!
matches('ab&#xA;cdefghij', '^cde', 'm'): It's a match!
```

Чтобы регулярное выражение работало так же, как в XML Schema, заключите его между метасимволами ^ и $. Выражение `^abc$` совпадает со строкой из трех символов `abc`, а выражение `abc` совпадает в строке любой длины, в которой три символа `abc` встречаются хотя бы один раз.

Обратные ссылки

Обратная ссылка позволяет получить доступ к совпадению предшествующего подвыражения *внутри самого регулярного выражения*. Допустим, вы хотите найти слова, которые начинаются и заканчиваются двумя одинаковыми буквами. Выражение

`([a-z]{2})(.*)\1` решает эту задачу. Запись `\1` представляет два символа, совпавших с первым подвыражением. Обратная ссылка `\2` представляет совпадение второго подвыражения, `\3` – третьего и т. д. (Выражения подсчитываются слева направо.)

Рассмотрим следующую таблицу стилей:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- back-reference.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>Using back-references to find words that begin &#xA;</xsl:text>
    <xsl:text>  and end with the same two letters:&#xA;&#xA;</xsl:text>
    <xsl:text>  replace($x, '([a-z]{2})(.*)\1', '$1--$2--$1')&#xA;</xsl:text>
    <xsl:text>    edited:    </xsl:text>
    <xsl:value-of
      select="replace('edited', '([a-z]{2})(.*)\1', '$1--$2--$1')"/>
    <xsl:text>&#xA;    editor:    </xsl:text>
    <xsl:value-of
      select="replace('editor', '([a-z]{2})(.*)\1', '$1--$2--$1')"/>
    <xsl:text>&#xA;    educated:  </xsl:text>
    <xsl:value-of
      select="replace('educated', '([a-z]{2})(.*)\1', '$1--$2--$1')"/>
    <xsl:text>&#xA;    orator:    </xsl:text>
    <xsl:value-of
      select="replace('orator', '([a-z]{2})(.*)\1', '$1--$2--$1')"/>
  </xsl:template>

</xsl:stylesheet>
```

Таблица стилей выдает следующий результат:

```
Using back-references to find words that begin
  and end with the same two letters:

  replace($x, '([a-z]{2})(.*)\1', '$1--$2--$1')
    edited:    ed--it--ed
    editor:    editor
    educated:  ed--ucat--ed
    orator:    or--at--or
```

Регулярное выражение совпадает со словами `edited`, `educated` и `orator`. Функция `replace()` записывает по две буквы в начале и конце слова, отделяя все, что находится между ними, двойными дефисами. В слове `editor` регулярное выражение не совпадает, поэтому это слово не изменяется.

Метасимволы

В регулярном выражении многие символы представляют сами себя. Например, буква `A` в регулярном выражении представляет прописную букву A. Конечно, существуют и специальные символы, которые обрабатываются по другим правилам:

. По умолчанию совпадает с любым символом, кроме символа новой строки. В режиме «совпадения точки со всеми символами» совпадает также с символом новой строки.

^ По умолчанию представляет начало строкового литерала. В многострочном режиме представляет начало внутренней «логической» строки.

[XPath] Используйте символ ^ для обозначения начала логической или физической строки в дополнение к синтаксису регулярных выражений, определяемому в XML Schema.

Также может использоваться в символьных классах для обозначения инверсии. Например, запись `[a-f]` представляет букву в диапазоне от `a` до `f`, а `[^a-f]` – любой символ, кроме букв в диапазоне от `a` до `f`.

$ *[XPath]* По умолчанию представляет конец строкового литерала. В многострочном режиме представляет конец внутренней «логической» строки.

\ Экранирует следующий символ (т. е. обеспечивает его литеральную интерпретацию).

| Альтернатива. Выражение `A|B` совпадает как с `A`, так и с `B`, но не с `AB`.

? Нуль или один экземпляр. Например, `A[A-Z]?Z` совпадает с `AZ` и `ABZ`, но не с `ABCZ`.

* Нуль или более экземпляров. Например, `A[A-Z]*` совпадает с любой строкой из прописных латинских букв, начинающейся с буквы A, за которой следует нуль или более прописных латинских букв. Выражение совпадает со строками `A`, `ABC`, `AA` и `AREALLYLONGSTRING`.

+ Один или более экземпляров. Например, `A[A-Z]+` совпадает с любой строкой из прописных латинских букв, начинающейся с буквы A, за которой следует одна или несколько прописных латинских букв. Выражение совпадает со строками `AA`, `AD` и `ADD`, но не с `A`.

(Начинает подвыражение. Например, в выражении `A(BCD)+` круглые скобки определяют подвыражение `BCD`, которое либо входит в одном экземпляре, либо не входит вообще. Выражение совпадает со строками `A` и `ABCD`.

) Завершает подвыражение.

[Начинает запись символьного класса. Например, выражение `[a-f]` обозначает букву в диапазоне от `a` до `f`.

- В квадратных скобках разделяет границы диапазона. Например, выражение `[a-f]` обозначает букву в диапазоне от `a` до `f`. Выражения `[a-f]` и `a|b|c|d|e|f` эквивалентны.

] Завершает символьный класс.

{ Начинает именованную категорию, блок или квантификатор. Например, `\p{IsThai}` обозначает набор всех тайских символов (от `&#x0E00` до `&#x0E7F`), а `\p{Lu}` обозначает все прописные буквы.

Квантификаторы могут иметь три формы: `{x}`, `{x,}` и `{x,y}`, где `x` и `y` – целые числа, причем `y` больше `x`. Квантификатор `{2}` обозначает ровно два экземпляра чего-либо, `{2,}` – не менее двух экземпляров, а `{2,5}` – от двух до пяти экземпляров. Выражение `(AB|AC|AD){2,5}` совпадает с `ACAB` и `ACADACADAB`, но не с `AB` или `ABABABABABAB`.

} Завершает именованную категорию, блок или квантификатор.

Поскольку фигурные скобки ({ и }) идентифицируют шаблоны значений атрибутов, в регулярных значениях внутри атрибутов необходимо использовать двойные фигурные скобки. Например, для поиска числа

из пяти цифр при помощи элемента <xsl:analyze-string> можно использовать следующую запись:

```
<xsl:analyze-string select="." regex="[0-9]{{5}}" ...>
```

Если не удвоить фигурные скобки внутри атрибута, регулярное выражение [0-9]{5} (с одиночными фигурными скобками) интерпретируется как одна цифра, за которой следует цифра 5.

С другой стороны, при использовании регулярных выражений при вызове функций фигурные скобки *не* удваиваются. Замена числа из пяти цифр дефисами может быть выполнена следующим образом:

```
<xsl:value-of select="replace(., [0-9]{5}, '-----')"/>
```

Комбинации для обозначения одиночных символов

В языке регулярных выражений предусмотрены служебные комбинации для обозначения некоторых специальных символов:

\n Символ новой строки (
).

\r Символ возврата курсора (
).

\t Символ табуляции (	).

\\ Обратная косая черта (\).

\| Вертикальная черта (|).

\. Точка (.).

\- Дефис или минус (-).

\^ Циркумфлекс (^).

\? Вопросительный знак (?).

* Звездочка (*).

\+ Плюс (+).

\{ Левая фигурная скобка ({).

\} Правая фигурная скобка (}).

\(Левая круглая скобка (().

\) Правая круглая скобка ()).

\[Левая квадратная скобка ([).

\] Правая квадратная скобка (]).

Комбинации для обозначения нескольких символов

Существуют также служебные комбинации для представления нескольких символов:

\s Любой пропуск.

\S Любой символ, не являющийся пропуском.

`\d` Любая цифра.

`\D` Любой символ, не являющийся цифрой.

`\w` Любой символ слова. «Символами слова» называются все символы, кроме знаков препинания, разделителей и других символов (символьные группы, определяемые с P, Z и C).

`\W` Любой символ, не являющийся символом слова.

`\i` Любой символ, который может быть первой буквой в имени XML. К этой категории относятся символ подчеркивания (_), двоеточие (:) и символы всех языков мира, определяемые как буквы в спецификации XML.

`\I` Любой символ, который не может быть первой буквой в имени XML.

`\c` Любой символ, который может использоваться в имени XML. К этой категории относятся все символы, которые могут быть первой буквой (`\i`), точка (.), дефис (-) и символы, определяемые как цифры.

`\C` Любой символ, который не может использоваться в имени XML.

`\p` Символ из следующей именованной группы. Например, `\p{Nd}` совпадает с «числовой»[1] цифрой, а `\p{IsThai}` – с символом тайского языка.

`\P` Любой символ, не принадлежащий к следующей именованной группе. Например, `\P{Nd}` совпадает с любым символом, кроме «числовой» цифры, а `\P{IsThai}` – с любым символом, кроме символа тайского языка.

Символьные группы

Символьные группы обозначают группы символов, объединенные общим признаком. Ввести один символ `L` проще, чем перечислять все возможные буквы, особенно если речь идет о многих тысячах символов, определенных в стандарте Юникод. Символьные группы используются с операторами `\p` и `\P`; запись `\p{L}` обозначает все буквы, а `\P{L}` – все символы, *кроме* букв. Символьные группы и их имена определяются в документе Unicode Character Database, доступном по адресу *http://www.unicode.org/Public/UNIDATA/UCD.html*.

Далее приведен список символьных групп.

Буквы

`L` Все буквы.

`Lu` Все буквы верхнего регистра.

`Ll` Все буквы нижнего регистра.

`Lt` Буквы титульного регистра, например символ `&#1C5;`, который состоит из прописной буквы D и строчной z с «птичкой» (caron).

`Lm` Модификаторы (например, символ `ª`, представляющий знак [ª]).

`Lo` Другие буквы, включая идеограммы, например буква «алеф» в иврите (`א`).

[1] В отличие от «буквенной», например шестнадцатеричной, цифры. – *Примеч. ред.*

Диакритика

`M` Все знаки.

`Mn` Непротяженные знаки (например, символ `́` – комбинационный восходящий акцент в Юникоде).

`Mc` Протяженные комбинационные знаки.

`Me` Огораживающие знаки.

Цифры

`N` Все цифры, включая альтернативные формы, такие как цифры, заключенные в круг (например, символ Юникода `①` – цифра 1 в круге), и цифры, заключенные в круглые скобки (например, символ Юникода `প` – цифра 1 в скобках).

`Nd` Десятичные цифры (в отличие, например, от `Ⅶ` – представление римской цифры VII в Юникоде).

`Nl` Буквенные цифры (в отличие от числовых; в частности, упоминавшийся ранее символ Юникода `Ⅶ` относится к этой категории).

`No` Другие числовые символы, например `½` – представление дроби ½ в виде одного символа.

Знаки препинания

`P` Все знаки препинания.

`Pc` Соединители, например символ подчеркивания `_`.

`Pd` Дефисы, например знак `-`.

`Ps` Открывающие (начальные) символы, например левая фигурная скобка ({).

`Pe` Закрывающие символы, например правая фигурная скобка (}).

`Pi` Начальные знаки цитирования. Учтите, что в некоторых языках начальный знак цитирования может входить в набор закрывающих символов (`Pe`). Например, в шведском языке цитирование начинается и завершается двойной закрывающей кавычкой (`ȝ`). В этом случае начальный знак цитирования не будет найдет при поиске по группе `Pi`.

`Pf` Конечные знаки цитирования. Как и в случае с начальными знаками цитирования, символ, используемый в качестве конечного знака цитирования в конкретном языке, может принадлежать набору открывающих или закрывающих символов.

`Po` Другие знаки препинания.

Разделители

`Z` Все разделители.

`Zs` Пробельные символы, например em-пробел (символ Юникода ` `) или en-пробел (символ Юникода ` `).

`Zl` Разделитель строк (символ Юникода ` `).

`Zp` Разделитель абзацев (символ Юникода ` `).

Символические знаки

`S` Все символические знаки.

`Sm` Математические знаки.

`Sc` Знаки денежных единиц.

`Sk` Модификаторы.

`So` Другие символические знаки.

Прочие символы

`C` Все остальные символы.

`Cc` Управляющие символы.

`Cf` Символы форматирования.

`Co` Приватное использование (зарезервировано в Юникоде).

`Cn` Не присвоено.

В стандарте Юникод определяется также класс «суррогатных» символов `Cs`. Эти символы не применимы к документам XML; к моменту разбора документа суррогатные символы уже не существуют. По этой причине `Cs` не является действительной символьной группой.

Блоки

Блочные служебные последовательности (блоки) представляют собой простой способ обозначения диапазонов символов, обладающих общим свойством. Каждый блок обладает именем, перед которым ставится префикс `Is`. Блоки используются операторами `\p` и `\P`. Например, выражение `\p{IsThai}` обозначает символы тайского языка (`&#x0E00 – &#x0E7F`). Выражение `\P{IsThai}` обозначает все символы, кроме тайских. Имена блоков перечислены далее в формате, определенном в спецификации XML Schema.

В табл. E.1 приведен полный список блоков. Таблица была сгенерирована на базе версии 5.0.0 файла *blocks.txt*. Список имен блоков является частью Unicode Character Database; последняя версия стандарта Юникод доступна по адресу *http://www.unicode.org/*.

Таблица E.1. Имена блоков

Имя блока	Начальный символ	Конечный символ
BasicLatin	&#x0000	&#x007F
Latin-1Supplement	&#x0080	&#x00FF
LatinExtended-A	&#x0100	&#x017F
LatinExtended-B	&#x0180	&#x024F
IPAExtensions	&#x0250	&#x02AF
SpacingModifierLetters	&#x02B0	&#x02FF
CombiningDiacriticalMarks	&#x0300	&#x036F

Имя блока	Начальный символ	Конечный символ
GreekandCoptic	&#x0370	&#x03FF
Cyrillic	&#x0400	&#x04FF
CyrillicSupplement	&#x0500	&#x052F
Armenian	&#x0530	&#x058F
Hebrew	&#x0590	&#x05FF
Arabic	&#x0600	&#x06FF
Syriac	&#x0700	&#x074F
ArabicSupplement	&#x0750	&#x077F
Thaana	&#x0780	&#x07BF
NKo	&#x07C0	&#x07FF
Devanagari	&#x0900	&#x097F
Bengali	&#x0980	&#x09FF
Gurmukhi	&#x0A00	&#x0A7F
Gujarati	&#x0A80	&#x0AFF
Oriya	&#x0B00	&#x0B7F
Tamil	&#x0B80	&#x0BFF
Telugu	&#x0C00	&#x0C7F
Kannada	&#x0C80	&#x0CFF
Malayalam	&#x0D00	&#x0D7F
Sinhala	&#x0D80	&#x0DFF
Thai	&#x0E00	&#x0E7F
Lao	&#x0E80	&#x0EFF
Tibetan	&#x0F00	&#x0FFF
Myanmar	&#x1000	&#x109F
Georgian	&#x10A0	&#x10FF
HangulJamo	&#x1100	&#x11FF
Ethiopic	&#x1200	&#x137F
EthiopicSupplement	&#x1380	&#x139F
Cherokee	&#x13A0	&#x13FF
UnifiedCanadianAboriginalSyllabics	&#x1400	&#x167F
Ogham	&#x1680	&#x169F
Runic	&#x16A0	&#x16FF
Tagalog	&#x1700	&#x171F
Hanunoo	&#x1720	&#x173F

Таблица Е.1 (продолжение)

Имя блока	Начальный символ	Конечный символ
Buhid	&#x1740	&#x175F
Tagbanwa	&#x1760	&#x177F
Khmer	&#x1780	&#x17FF
Mongolian	&#x1800	&#x18AF
Limbu	&#x1900	&#x194F
TaiLe	&#x1950	&#x197F
NewTaiLue	&#x1980	&#x19DF
KhmerSymbols	&#x19E0	&#x19FF
Buginese	&#x1A00	&#x1A1F
Balinese	&#x1B00	&#x1B7F
PhoneticExtensions	&#x1D00	&#x1D7F
PhoneticExtensionsSupplement	&#x1D80	&#x1DBF
CombiningDiacriticalMarksSupplement	&#x1DC0	&#x1DFF
LatinExtendedAdditional	&#x1E00	&#x1EFF
GreekExtended	&#x1F00	&#x1FFF
GeneralPunctuation		&#x206F
SuperscriptsandSubscripts	&#x2070	&#x209F
CurrencySymbols	&#x20A0	&#x20CF
CombiningDiacriticalMarksforSymbols	&#x20D0	&#x20FF
LetterlikeSymbols	&#x2100	&#x214F
NumberForms	&#x2150	&#x218F
Arrows	&#x2190	&#x21FF
MathematicalOperators	&#x2200	&#x22FF
MiscellaneousTechnical	&#x2300	&#x23FF
ControlPictures	&#x2400	&#x243F
OpticalCharacterRecognition	&#x2440	&#x245F
EnclosedAlphanumerics	&#x2460	&#x24FF
BoxDrawing	&#x2500	&#x257F
BlockElements	&#x2580	&#x259F
GeometricShapes	&#x25A0	&#x25FF
MiscellaneousSymbols	&#x2600	&#x26FF
Dingbats	&#x2700	&#x27BF
MiscellaneousMathematicalSymbols-A	&#x27C0	&#x27EF

Имя блока	Начальный символ	Конечный символ
SupplementalArrows-A	&#x27F0	&#x27FF
BraillePatterns	&#x2800	&#x28FF
SupplementalArrows-B	&#x2900	&#x297F
MiscellaneousMathematicalSymbols-B	&#x2980	&#x29FF
SupplementalMathematicalOperators	&#x2A00	&#x2AFF
MiscellaneousSymbolsandArrows	&#x2B00	&#x2BFF
Glagolitic	&#x2C00	&#x2C5F
LatinExtended-C	&#x2C60	&#x2C7F
Coptic	&#x2C80	&#x2CFF
GeorgianSupplement	&#x2D00	&#x2D2F
Tifinagh	&#x2D30	&#x2D7F
EthiopicExtended	&#x2D80	&#x2DDF
SupplementalPunctuation	&#x2E00	&#x2E7F
CJKRadicalsSupplement	&#x2E80	&#x2EFF
KangxiRadicals	&#x2F00	&#x2FDF
IdeographicDescriptionCharacters	&#x2FF0	&#x2FFF
CJKSymbolsandPunctuation		&#x303F
Hiragana	&#x3040	&#x309F
Katakana	&#x30A0	&#x30FF
Bopomofo	&#x3100	&#x312F
HangulCompatibilityJamo	&#x3130	&#x318F
Kanbun	&#x3190	&#x319F
BopomofoExtended	&#x31A0	&#x31BF
CJKStrokes	&#x31C0	&#x31EF
KatakanaPhoneticExtensions	&#x31F0	&#x31FF
EnclosedCJKLettersandMonths	&#x3200	&#x32FF
CJKCompatibility	&#x3300	&#x33FF
CJKUnifiedIdeographsExtensionA	&#x3400	&#x4DBF
YijingHexagramSymbols	&#x4DC0	&#x4DFF
CJKUnifiedIdeographs	&#x4E00	&#x9FFF
YiSyllables	&#xA000	&#xA48F
YiRadicals	&#xA490	&#xA4CF
ModifierToneLetters	&#xA700	&#xA71F
LatinExtended-D	&#xA720	&#xA7FF

Таблица Е.1 (продолжение)

Имя блока	Начальный символ	Конечный символ
SylotiNagri	&#xA800	&#xA82F
Phags-pa	&#xA840	&#xA87F
HangulSyllables	&#xAC00	&#xD7AF
HighSurrogates	&#xD800	&#xDB7F
HighPrivateUseSurrogates	&#xDB80	&#xDBFF
LowSurrogates	&#xDC00	&#xDFFF
PrivateUseArea	&#xE000	&#xF8FF
CJKCompatibilityIdeographs	&#xF900	&#xFAFF
AlphabeticPresentationForms	&#xFB00	&#xFB4F
ArabicPresentationForms-A	&#xFB50	&#xFDFF
VariationSelectors	&#xFE00	&#xFE0F
VerticalForms	&#xFE10	&#xFE1F
CombiningHalfMarks	&#xFE20	&#xFE2F
CJKCompatibilityForms	&#xFE30	&#xFE4F
SmallFormVariants	&#xFE50	&#xFE6F
ArabicPresentationForms-B	&#xFE70	&#xFEFF
HalfwidthandFullwidthForms	&#xFF00	&#xFFEF
Specials	&#xFFF0	&#xFFFF
LinearBSyllabary	&#x10000	&#x1007F
LinearBIdeograms	&#x10080	&#x100FF
AegeanNumbers	&#x10100	&#x1013F
AncientGreekNumbers	&#x10140	&#x1018F
OldItalic	&#x10300	&#x1032F
Gothic	&#x10330	&#x1034F
Ugaritic	&#x10380	&#x1039F
OldPersian	&#x103A0	&#x103DF
Deseret	&#x10400	&#x1044F
Shavian	&#x10450	&#x1047F
Osmanya	&#x10480	&#x104AF
CypriotSyllabary	&#x10800	&#x1083F
Phoenician	&#x10900	&#x1091F
Kharoshthi	&#x10A00	&#x10A5F
Cuneiform	&#x12000	&#x123FF

Имя блока	Начальный символ	Конечный символ
CuneiformNumbersandPunctuation	&#x12400	&#x1247F
ByzantineMusicalSymbols	&#x1D000	&#x1D0FF
MusicalSymbols	&#x1D100	&#x1D1FF
AncientGreekMusicalNotation	&#x1D200	&#x1D24F
TaiXuanJingSymbols	&#x1D300	&#x1D35F
CountingRodNumerals	&#x1D360	&#x1D37F
MathematicalAlphanumericSymbols	&#x1D400	&#x1D7FF
CJKUnifiedIdeographsExtensionB	&#x20000	&#x2A6DF
CJKCompatibilityIdeographsSupplement	&#x2F800	&#x2FA1F
Tags	&#xE0000	&#xE007F
VariationSelectorsSupplement	&#xE0100	&#xE01EF
SupplementaryPrivateUseArea-A	&#xF0000	&#xFFFFF
SupplementaryPrivateUseArea-B	&#x100000	&#x10FFFF

Последнюю версию списка имен блоков и содержащихся в них символов можно найти в файле *blocks.txt* на сайте *http://www.unicode.org/*.

F

Форматные коды XSLT

В этом приложении перечисляются коды форматирования чисел, дат и времени, применяемые в XSLT. Эти коды используются следующими элементами и функциями:

- Функции `format-date()`, `format-time()` и `format-dateTime()`, определяемые в XSLT 2.0, раздел 16.5 «Formatting Dates and Times».
- Атрибут `format` элемента `<xsl:number>`, определяемый в XSLT 1.0, раздел 7.7 «Numbering». (Возможности атрибута `format` описаны в спецификации XSLT 2.0, раздел 12 «Numbering».)
- Элемент `<xsl:decimal-format>`, определяемый в XSLT 1.0, раздел 12.3 «Number Formatting».
- Функция `format-number()`, определяемая в XSLT 1.0 и XSLT 2.0.

Большинство кодов форматирования чисел было определено в XSLT 1.0. Основные дополнения к возможностям числового форматирования, появившиеся в XSLT 2.0, относились к поддержке числительных и других языков. В XSLT 2.0 были добавлены функции форматирования даты и времени. Существуют три разновидности форматных кодов:

- Форматные коды даты и времени, определяемые стандартом ISO 8601 для представления даты и времени. Эти форматные коды используются для создания новых значений `xs:date`, `xs:time`, `xs:dateTime`, `xs:duration`, `xs:dayTimeDuration` и `xs:yearMonthDuration`.
- Форматные коды даты и времени, определяемые в XSLT 2.0, раздел 16.5 «Formatting Dates and Times». Используются для форматирования частей значений `xs:date`, `xs:time`, `xs:dateTime`, `xs:duration`, `xs:dayTimeDuration` и `xs:yearMonthDuration`.
- Коды форматирования нумерации, определяемые в XSLT 1.0, раздел 7.7 «Numbering» и раздел 12.3 «Number Formatting». Эти коды были дополнены в XSLT 2.0, раздел 12 «Numbering» и раздел 16.4 «Number Formatting».

Коды форматирования чисел

В следующих подразделах более подробно рассматриваются возможности форматирования чисел. Они используются элементами <xsl:decimal-format> и <xsl:number>, а также функцией format-number().

Компоненты чисел

В табл. F.1 описываются символы, представляющие компоненты числа. Они используются функцией format-number() и элементом <xsl:decimal-format>.

Таблица F.1. Коды форматирования чисел

Код	Описание
#	Цифра. Начальные и конечные нули не отображаются. Форматирование числа 4.0 по строке "#.##" возвращает строку "4".
0	Цифра. В отличие от символа #, отображает начальные и конечные нули. Форматирование числа 4.1 по строке "#.00" возвращает строку "4.10".
.	Десятичная точка (используется для отделения дробной части).
-	Знак «минус».
,	Разделитель групп разрядов.
;	Разделитель шаблонов для положительного и отрицательного чисел.
%	Признак отображения числа в процентах. Значение умножается на 100 и выводится со знаком процента. Форматирование числа .76 по строке "##%" возвращает строку "76%".
\u2030	Символ Юникода для обозначения промилле. Значение умножается на 1000 и выводится со знаком промилле. Форматирование числа .768 по строке "###\2030" возвращает строку "768‰".

Компоненты десятичных форматов

Элемент <xsl:decimal-format> позволяет определять именованные форматы для десятичных чисел. Каждый именованный формат представляет собой группу параметров со свойствами, перечисленными в табл. F.2.

Таблица F.2. Компоненты десятичных форматов

Имя компонента	Описание	Значение по умолчанию
name	Имя формата.	нет
decimal-separator	Символ (обычно точка или запятая), используемый для отделения дробной части. Используется как в форматной строке, так и в выводе.	. (точка)
grouping-separator	Символ (обычно точка или запятая), используемый в качестве разделителя групп разрядов. Используется как в форматной строке, так и в выводе.	, (запятая)

Таблица F.2 (продолжение)

Имя компонента	Описание	Значение по умолчанию
infinity	Строка для представления бесконечности. Учтите, что в вычислениях XSLT поддерживаются как положительная, так и отрицательная бесконечности. Строка используется только в выводе.	Infinity
minus-sign	Символ, применяемый в качестве знака «минус». Используется только в выводе.	- (дефис, -)
NaN	Строка, отображаемая в том случае, если форматируемое значение не является числом. Используется только в выводе.	NaN
percent	Символ для обозначения процентов. Используется как в форматной строке, так и в выводе.	% (процент)
per-mille	Символ для обозначения промилле. Используется как в форматной строке, так и в выводе.	‰ (символ промилле в Юникоде, ‰).
zero-digit	Символ для обозначения цифры «нуль». Используется как в форматной строке, так и в выводе.	0 (нуль)
digit	Символ для обозначения цифры в форматной строке.	#
pattern-separator	Символ для разделения подшаблонов положительного и отрицательного представлений числа. Используется только в форматной строке.	; (точка с запятой)

Коды форматирования даты и времени

Компоненты даты и времени

В табл. F.3 описаны коды, определенные для разных частей значений xs:date, xs:dateTime и xs:time.

Таблица F.3. Компоненты даты и времени

Спецификатор	Описание	Модификатор представления по умолчанию
Y	Год (абсолютное значение)	1
M	Месяц года	1
D	День месяца	1
d	День года	1
F	День недели	n

Спецификатор	Описание	Модификатор представления по умолчанию
W	Неделя года	1
w	Неделя месяца	1
H	Час суток (24 часа)	1
h	Час половины суток (12 часов)	1
p	Маркер a.m./p.m.	n
m	Минуты	01
s	Секунды	01
f	Дробные секунды	1
Z	Часовой пояс, заданный смещением от UTC, или при наличии алфавитного модификатора традиционное имя часового пояса (например, PST).	1
z	Часовой пояс, заданный смещением от GMT, например GMT+1	1
C	Календарь: название или сокращение календаря	n
E	Эра: название точки отсчета лет (например, годы правления монарха)	n

Модификаторы представления

Модификатором представления называется код, определяющий способ отображения части значения даты или времени. Например, модификатор FNn выводит день недели в виде строки, записанной с прописной буквы с использованием текущего языка. Если в системе используется английский язык и календарь, принятый в США, для 15 августа 2007 года названием дня недели (FNn) является строка Wednesday. С кодом форматирования FWw выводится числовое значение Four. В системе с немецким языком те же коды с той же датой выводят значения Mittwoch и Drei (в европейском календаре среда является третьим днем недели). Модификаторы представления перечислены в табл. F.4.

Таблица F.4. Модификаторы представления

Модификатор	Описание
1	Значение выводится в виде числа из одной или более цифр: 0, 1, 2, 3, 4, ...
01	Значение выводится в виде числа из двух или более цифр: 00, 01, 02, ..., 09, 10, 11, ..., 99, 100, 101, ...
(другие числовые значения)	Процессор XSLT может поддерживать другие варианты числовых представлений. Пример из спецификации: код форматирования ๑ приказывает процессору использовать тайскую нумерацию. За информацией о поддерживаемых числовых представлениях обращайтесь к документации своего процессора.

Таблица F.4 (продолжение)

Модификатор	Описание
A	Значение выводится буквами верхнего регистра: A, B, C, ..., Z, AA, AB, AC, ...
a	Значение выводится буквами нижнего регистра: a, b, c, ..., z, aa, ab, ac, ...
I	Значение выводится римскими цифрами верхнего регистра: I, II, III, IV, V, VI, VII, VIII, IX, X, ...
i	Значение выводится римскими цифрами нижнего регистра: i, ii, iii, iv, v, vi, vii, viii, ix, x, ...
n	Название компонента в нижнем регистре. Модификатор применим только к компонентам, обладающим названием, например к дню недели (F), месяцу года (M) и часовому поясу (Z).
N	Название компонента в верхнем регистре. Модификатор применим только к компонентам, обладающим названием, например к дню недели (F), месяцу года (M) и часовому поясу (Z).
Nn	Название компонента в титульном регистре. Модификатор применим только к компонентам, обладающим названием, например к дню недели (F), месяцу года (M) и часовому поясу (Z).
w	Значение выводится в виде слова, записанного в нижнем регистре: one, two, three, four, ...
W	Значение выводится в виде слова, записанного в верхнем регистре: ONE, TWO, THREE, FOUR, ...
Ww	Значение выводится в виде слова, записанного с прописной буквы: One, Two, Three, Four, ...

Календари

Далее приводятся названия календарей, определяемые в спецификации XSLT 2.0. Кроме значений, перечисленных в табл. F.5, процессоры XSLT могут поддерживать другие календари. Сокращенные названия календарей, не включенных в таблицу, должны уточняться префиксом пространства имен. За информацией о поддерживаемых календарях обращайтесь к документации процессора XSLT.

Таблица F.5. Календари

Сокращение	Календарь
AD	Anno Domini (Христианская эра)
AH	Anno Hegirae (мусульманский календарь)
AME	Mauludi Era (солнечные годы, начиная с дня рождения Мухаммеда)
AM	Anno Mundi (еврейский календарь)
AP	Anno Persici (персидский календарь)
AS	Aji Saka Era (календарь о. Ява)

Сокращение	Календарь
BE	Buddhist Era (буддистский календарь)
CB	Cooch Behar Era (индийский календарь)
CE	Common Era (наша эра)
CL	Chinese Lunar Era (китайский календарь)
CS	Chula Sakarat Era (таиландский календарь)
EE	Ethiopian Era (эфиопский календарь)
FE	Fasli Era (индийский календарь)
ISO	Календарь ISO 8601
JE	Japanese Calendar (японский календарь)
KE	Khalsa Era (сикхский календарь)
KY	Kali Yuga (индусский календарь)
ME	Malabar Era (малабарский календарь)
MS	Monarchic Solar Era (иранский календарь)
NS	Nepal Samwat Era (непальский календарь)
OS	Old Style (юлианский календарь)
RS	Rattanakosin Era (календарь Бангкока)
SE	Saka Era (индийский календарь)
SH	Mohammedan Solar Era (иранский календарь)
SS	Saka Samvat (индийский календарь)
TE	Tripurabda Era (индийский календарь)
VE	Vikrama Era (индийский календарь)
VS	Vikrama Samvat Era (индийский календарь)

G

Руководство по переходу на XSLT 2.0

В этом приложении даются некоторые полезные советы и рекомендации по переводу таблиц стилей XSLT 1.0 на XSLT 2.0. Большинство таблиц стилей XSLT 1.0 нормально работает в процессорах XSLT 2.0, однако существуют некоторые ошибки, которых следует остерегаться. Также мы рассмотрим некоторые новые возможности XSLT 2.0. Если ни одна из новых возможностей XSLT 2.0 и XPath 2.0 для вас не представляет ценности, реальной необходимости переводить на них свои таблицы стилей нет. (Крайне маловероятно, чтобы ничто в XSLT 2.0 и XPath 2.0 вам не пригодилось, но все же...)

Новые полезные возможности XSLT 2.0 и XPath 2.0

Мы обсудим самые важные новые возможности XSLT 2.0 и XPath 2.0. Список не полон, но он дает представление о том, в какой степени XSLT 2.0 упрощает таблицы стилей и приложения.

Рекурсия применяется значительно реже

В таблицах стилей XSLT 1.0 рекурсия была стандартным приемом. Например, для выполнения замены в строках приходилось писать рекурсивный шаблон, который заменял первый экземпляр подстроки, а затем вызывал себя с оставшейся частью строки. В XSLT 2.0 можно воспользоваться функцией `replace()`. Рекурсивные шаблоны создают массу проблем с написанием и сопровождением; если их удается заменить более простым кодом в таблице стилей XSLT 2.0, дело того стоит. За подробностями обращайтесь к разделу «[2.0] Использование функции XPath 2.0 replace() для предотвращения рекурсии» главы 5. Другой хороший пример приведен в разделе «Вычисления без рекурсии» главы 8.

Значительное упрощение группировки

В XSLT 2.0 группировка реализуется на порядок проще, чем в XSLT 1.0. Элемент `<xsl:for-each-group>` сокращает объем кода до 80% в зависимости от сложности таблиц стилей XSLT 1.0. Если в существующих таблицах стилей выполняется группи-

ровка, с новыми возможностями XSLT 2.0 решение становится намного более элегантным. За подробностями обращайтесь к разделу «[2.0] Новый синтаксис группировки в XSLT 2.0» главы 7.

Поддержка типов данных и XML Schema

Если проверка типов и данных играет важную роль в вашем приложении, переход на XSLT 2.0 абсолютно необходим. Базовый процессор XSLT 2.0 поддерживает все базовые типы данных XML Schema, а схемосовместимый процессор позволяет определять собственные типы данных и использовать их в таблицах стилей. Типы данных и поддержка схем рассматриваются в главе 3.

Поддержка регулярных выражений

XSLT 1.0 поддерживает такие функции, как `contains()`, `substring-before()` и `substring-after()`, но они не могут рассматриваться как замена регулярных выражений. Поддержка регулярных выражений в XSLT 2.0 отличается исключительной гибкостью и мощью. Подробнее поддержка регулярных выражений обсуждается в описаниях элемента *[2.0]* `<xsl:analyze-string>` и функций *[2.0]* `matches()`, *[2.0]* `replace()` и *[2.0]* `tokenize()`. В приложении E приводится более полный обзор синтаксиса регулярных выражений, используемого в XSLT 2.0 и XPath 2.0.

Потенциальные ошибки

Между XSLT 1.0 и XSLT 2.0 существует ряд различий, которые могут стать причиной фатальных ошибок. Если вы собираетесь переносить свои таблицы стилей с версии 1.0 на версию 2.0, вам придется учитывать все эти различия.

Ошибка при передаче неопределенных параметров <xsl:call-template>

В XSLT 1.0 при использовании элемента `<xsl:call-template>` для вызова шаблона можно было передать любое количество параметров. Если какие-либо параметры не были определены в шаблоне, они игнорировались. В XSLT 2.0 вызов `<xsl:call-template>` с неопределенным параметром приводит к фатальной ошибке.

Передача неопределенных параметров элементам `<xsl:apply-templates>`, `<xsl:apply-imports>` или `<xsl:next-match>` ***не является*** ошибкой.

В таблицах стилей XSLT 1.0 лишние параметры часто передавались из-за того, что когда-то в будущем они могли понадобиться другому шаблону. При каждом вызове шаблона (посредством `<xsl:apply-templates>` или `<xsl:call-template>`) передавались лишние параметры. В XSLT 2.0 появился намного более удобный механизм *туннельных параметров*. Туннельный параметр не передается каждому промежуточному шаблону, благодаря чему таблицы стилей становятся более логичными и понятными. За дополнительной информацией обращайтесь к разделу «Туннельные параметры» главы 5.

Эта потенциальная проблема относится к статическим ошибкам. Иначе говоря, процессор XSLT 2.0 может выявить передачу неверного количества параметров шаблону без фактического преобразования документа XML.

Различия при выполнении вычислений

В XSLT 1.0 для операций с числами использовался простой тип данных `number`. Математические операции, особенно связанные с делением на нуль, в XSLT 2.0 работают иначе.

Ситуация осложняется тем, что поведение XSLT 2.0 изменяется в зависимости от числового типа. Деление `xs:integer` или `xs:decimal` на нуль является фатальной ошибкой, а деление `xs:double` или `xs:float` возвращает `INF` (бесконечность).

Самый простой способ избежать здесь ошибки – включить в элемент атрибут `version="1.0"`. Это заставит процессор XSLT 2.0 обработать выражение по правилам XSLT 1.0, и ваша таблица стилей будет работать так, как работала всегда.

```
<xsl:variable version="1.0" name="ratio"
  select="$orders div $returns"/>
```

Другое, более элегантное решение основано на использовании нового оператора XPath `if`. Например, код может выглядеть так:

```
<xsl:variable name="ratio"
  select="if ($returns != 0) then
          $orders div $returns else
          0"/>
```

Если значение `$returns` отлично от нуля, вычисления выполняются, в противном случае возвращается нуль.

Деление работает с шестью типами данных: `xs:integer`, `xs:decimal`, `xs:float`, `xs:double`, `xs:yearMonthDuration` и `xs:dayTimeDuration`. Следующая таблица стилей показывает, что происходит при попытке деления на нуль:

```
<?xml version="1.0"?>
<!-- divide-by-zero.xsl -->
<xsl:stylesheet version="2.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>&#xA;Division by zero in XSLT 2.0:&#xA;&#xA;</xsl:text>

    <xsl:text>&lt;xsl:value-of select="1 div 0"/&gt;&#xA;</xsl:text>
    <xsl:text>  [fatal error]&#xA;&#xA;</xsl:text>
    <xsl:text>&lt;xsl:value-of select="1.0 div 0.0"/&gt;&#xA;</xsl:text>
    <xsl:text>  [fatal error]&#xA;&#xA;</xsl:text>
    <xsl:text>&lt;xsl:value-of select="xs:integer(1) </xsl:text>
    <xsl:text>div xs:integer(0)"/&gt;&#xA;</xsl:text>
    <xsl:text>  [fatal error]&#xA;&#xA;</xsl:text>
    <xsl:text>&lt;xsl:value-of select="xs:decimal(1.0) </xsl:text>
    <xsl:text>div xs:decimal(0.0)"/&gt;&#xA;</xsl:text>
    <xsl:text>  [fatal error]&#xA;&#xA;</xsl:text>
    <xsl:text>&lt;xsl:value-of select="xs:double(1.0) </xsl:text>
    <xsl:text>div xs:double(0.0)"/&gt;&#xA;  </xsl:text>
    <xsl:value-of
      select="(xs:double(1.0) div xs:double(0.0), '&#xA;&#xA;')"/>
    <xsl:text>&lt;xsl:value-of select="xs:float(1.0) </xsl:text>
    <xsl:text>div xs:float(0.0)"/&gt;&#xA;  </xsl:text>
```

```
    <xsl:value-of
      select="(xs:float(1.0) div xs:float(0.0), '&#xA;&#xA;')"/>

    <xsl:text>Dividing durations by zero:&#xA;&#xA;</xsl:text>
    <xsl:text>&lt;xsl:value-of select="xs:yearMonthDuration</xsl:text>
    <xsl:text>('P3Y4M') div xs:double(0.0)"/&gt;&#xA; </xsl:text>
    <xsl:text>  [fatal error]&#xA;&#xA;</xsl:text>
    <xsl:text>&lt;xsl:value-of select="xs:yearMonthDuration</xsl:text>
    <xsl:text>('P3Y4M') div &#xA;                      </xsl:text>
    <xsl:text>xs:yearMonthDuration('POYOM')"/&gt;&#xA; </xsl:text>
    <xsl:text>  [fatal error]&#xA;&#xA;</xsl:text>
    <xsl:text>&lt;xsl:value-of select="xs:dayTimeDuration</xsl:text>
    <xsl:text>('P3DT8H') div xs:double(0.0)"/&gt;&#xA; </xsl:text>
    <xsl:text>  [fatal error]&#xA;&#xA;</xsl:text>
    <xsl:text>&lt;xsl:value-of select="xs:dayTimeDuration</xsl:text>
    <xsl:text>('P3DT8H') div &#xA;                      </xsl:text>
    <xsl:text>xs:dayTimeDuration('PODTOH')"/&gt;&#xA; </xsl:text>
    <xsl:text>  [fatal error]&#xA;&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Как упоминалось ранее, при делении `xs:double` или `xs:float` на нуль возвращается `INF`; во всех остальных случаях происходит фатальная ошибка. Деление `xs:yearMonthDuration` или `xs:dayTimeDuration` на нуль или на интервал нулевой длины также приводит к фатальной ошибке.

Деление на нуль может быть распознано как статическая ошибка, но с гораздо большей вероятностью оно произойдет во время выполнения. Если вы адаптируете свою таблицу стилей для использования математических правил XSLT 2.0, протестируйте свой код и убедитесь в том, что все возможные ошибки деления на нуль обрабатываются правильно.

Ужесточение проверки типов

В XSLT 1.0 использовалась очень простая система типов. Поддержка XML Schema в XSLT 2.0 означает, что проверка типов стала намного строже. Например, в XSLT 1.0 следующая команда работает:

```
<xsl:value-of select="substring-before(12345, 3)"/>
```

и возвращает значение `12`, а в XSLT 2.0 происходит статическая ошибка. Чтобы эта функция работала в XSLT 2.0, ее необходимо записать следующим образом:

```
<xsl:value-of select="substring-before(string(12345), string(3))"/>
```

При переносе таблиц стилей XSLT 1.0 на 2.0 подобные явные преобразования типов приходится выполнять довольно часто.

Ошибка при вызове некоторых функций с несколькими узлами

В XSLT 1.0 некоторым функциям в аргументе передавался узел. Если функции передавалось более одного узла, функция просто брала первый узел и игнорировала все остальные. XSLT 2.0 действует гораздо жестче; передача нескольких узлов функции, которая должна получать один узел, приводит к фатальной ошибке. Конечно, можно сказать, что использование аргумента `/purchase-order/items/item` там, где име-

ется в виду /purchase-order/items/item[1], свидетельствует о неряшливом программировании. Согласны вы с этим мнением или нет, XSLT 2.0 заставляет передавать функции ровно один узел.

Функции XSLT 1.0 и XPath 1.0 generate-id(), local-name(), name(), namespace-uri(), number() и string() в XSLT 2.0 должны получать один узел. Чтобы избежать каких-либо проблем, достаточно включить предикат [1] в таблицы стилей в процессе переноса. Кроме того, если вы используете выражение XPath для выбора узлов, передаваемых функции concat(), это выражение не может возвращать более одного узла. Выражение concat('#1. ', /report/brand/name) приводит к ошибке в XSLT 2.0, если выражению XPath соответствует более одного узла. С другой стороны, concat('#1. ', (/report/brand/name)[1]) работает всегда.

Эта ошибка является динамической. Если выражения XPath обычно возвращают только один узел, обнаружить ее будет особенно трудно: ошибка возникает в редком случае, когда выражение XPath возвращает более одного узла. Если вам нужен только один узел, лучше всего заменить выражение x/y/z на (x/y/z)[1].

Способы перехода

Далее кратко перечисляются основные способы переноса таблиц стилей на платформу XSLT 2.0.

Написать таблицы стилей «с нуля»

Естественно, это самый радикальный вариант. Если кодовая база XSLT 1.0 имеет умеренный объем, можно перейти на написание таблиц стилей XSLT 2.0 «с нуля». Однако с большей вероятностью вы начнете с замены нескольких таблиц стилей, используемых в разных частях приложения, после чего воспользуетесь полученными знаниями и опытом для написания более крупных и сложных таблиц.

Этот способ требует значительных вложений в обучение и ресурсы разработки – понимания всех новых возможностей XSLT 2.0 и умения ими пользоваться.

Переключиться на версию 2.0 и посмотреть, что произойдет

В простейшем варианте перехода вы просто меняете атрибут version элемента <xsl:stylesheet> на 2.0. Если при этом возникнут какие-либо статические ошибки, процессор XSLT 2.0 немедленно обнаружит их. Либо исправьте эти ошибки, либо включите атрибут version="1.0" в отдельные элементы XSLT, чтобы процессор XSLT 2.0 использовал режим обратной совместимости.

При использовании этого подхода очень важно организовать проверку динамических ошибок, о которых говорилось в предыдущем разделе. Если вы включите атрибут version="2.0" в свою таблицу стилей и она будет работать с вашим документом XML, это не значит, что она будет работать с каждым документом. Как и при любых модификациях кода, очень важно провести тщательное тестирование, прежде чем запускать таблицу стилей в реальную эксплуатацию.

Заменить громоздкий код XSLT 1.0 новыми возможностями XSLT 2.0

В другом, менее радикальном способе перехода на XSLT 2.0 вы ищете в своих таблицах стилей шаблоны, которые можно заменить функциями XSLT 2.0. Например, если в таблице стилей используется рекурсивный шаблон для замены строк, замена его функцией `replace()` упростит код (и, скорее всего, код будет работать быстрее). Все, что использует рекурсии или функции расширения, является хорошим кандидатом для замены средствами XSLT 2.0.

Также обратите особое внимание на чрезмерное использование переменных. Если переменная применяется для хранения результата, сгенерированного именованным шаблоном, рассмотрите возможность замены именованного шаблона элементом `<xsl:function>`. Использование функции XSLT 2.0 означает, что код можно вызывать из выражения XPath. Вызов функции заменяет переменную с именованным шаблоном одной операцией.

Если шаблонам в таблице стилей передается большое количество параметров, рассмотрите возможность применения туннельных параметров. С ростом сложности и функциональности таблиц стилей шаблоны все чаще получают один или несколько параметров, которые затем передаются всем шаблонам, вызываемым из него. Таблице стилей приходится передавать эти параметры каждому шаблону на случай, если они понадобятся где-то еще. Туннельные параметры делают код более стройным и упрощают сопровождение.

Смешать код XSLT 1.0 и XSLT 2.0 в одной таблице стилей

Очень полезная методика перехода на XSLT 2.0 основана на обработке разных частей таблицы стилей в режиме XSLT 1.0 или XSLT 2.0. Простейший способ основан на создании таблицы стилей с атрибутом `version="2.0"` и добавлением атрибута `version="1.0"` в шаблоны по мере надобности. (Напомню, что атрибут `version` может использоваться в любом элементе XSLT, кроме `<xsl:output>`.) Это позволит вам организовать поэтапный перенос таблиц стилей и не беспокоиться о динамических ошибках. Если таблица стилей зависит от выполнения математических вычислений по правилам XSLT 1.0, настройте ее для использования `version="1.0"`. По мере накопления опыта работы на XSLT 2.0 вы сможете брать различные разделы таблицы стилей XSLT 1.0 и переносить их на XSLT 2.0.

Вообще не переходить на XSLT 2.0

Самое простое и быстрое решение. Если в ваших таблицах стилей новые возможности XSLT 2.0 не принесут ощутимой пользы, реальных причин для перехода нет. И все же крайне маловероятно, чтобы сколько-нибудь нетривиальную таблицу стилей нельзя было усовершенствовать применением новых возможностей.

Архитектура вашего приложения может помочь определить, нужно ли переходить на XSLT 2.0. Например, если приложение отправляет броузеру данные XML и таблицу стилей XSLT, броузер должен иметь XSLT 2.0-совместимый процессор. На момент написания книги броузеров со встроенной поддержкой XSLT 2.0 не существовало, поэтому в этой ситуации с переходом можно не торопиться.

Глоссарий

CDATA, секция

Секция документа XML, в которой игнорируется вся разметка. Секция CDATA начинается с символов `<![CDATA[` и завершается символами `]]>`. Если две правые квадратные скобки встречаются в содержимом секции CDATA, их необходимо экранировать. В таблице стилей невозможно определить, был ли некоторый текстовый узел изначально оформлен в виде секции CDATA. Некоторые элементы могут генерироваться в виде секций CDATA при помощи атрибута `cdata-section-elements` элемента `<xsl:output>`.

ID

Один из базовых типов данных, определяемых спецификацией XML. (Также определяется как тип `xs:ID` в XML Schema.) В документе XML один атрибут элемента может быть объявлен с типом `ID`; это означает, что значение данного атрибута должно быть уникально среди всех атрибутов типа `ID` всех элементов в документе. Каждый конкретный элемент может иметь не более одного атрибута с типом `ID`. Атрибуты типа `ID` могут использоваться для построения перекрестных ссылок функцией `id()`. За подробным описанием типов данных `ID`, `IDREF` и `IDREFS` обращайтесь к главе 5.

IDREF

Один из базовых типов данных, определяемых в спецификации XML. (Также определяется как тип `xs:IDREF` в XML Schema.) В документе XML атрибут, объявленный с типом `IDREF`, должен иметь значение, совпадающее со значением атрибута `ID` в другой точке документа. Атрибуты типа `IDREFS` могут использоваться для построения перекрестных ссылок функцией `id()`. За подробным описанием типов данных `ID`, `IDREF` и `IDREFS` обращайтесь к главе 5.

IDREFS

Один из базовых типов данных, определяемых в спецификации XML. (Также определяется как тип `xs:IDREFS` в XML Schema.) В документе XML атрибут, объявленный с типом `IDREFS`, должен содержать одно или несколько значений, разделенных пропусками, каждое из которых совпадает со значением атрибута `ID` в другой точке документа. Атрибуты типа `IDREFS` могут использоваться для построения перекрестных ссылок функцией `id()`. За подробным описанием типов данных `ID`, `IDREF` и `IDREFS` обращайтесь к главе 5.

NCName

Имя XML, не разделенное двоеточием, используемое для локальных имен и префиксов пространств имен. Должно начинаться с буквы или символа подчеркивания (_).

QName

Сокращение от *полное имя* (*qualified name*).

URI

Универсальный идентификатор ресурса (Universal Resource Identifier) – обобщенная форма URL-адресов, используемых в Web. URI-адреса определяются в документе RFC 2396 и поддерживают примерно 60 ASCII-символов. Интернационализированные идентификаторы ресурсов (IRI, Internationalized Resource Identifier), определяемые в RFC 3897, поддерживают (в XSLT 2.0) значительно больший диапазон символов. Вспомогательная функция XPath 2.0 `iri-to-uri()` преобразует IRI в URI.

URI-адрес пространства имен

URI-адрес, связанный с набором имен элементов и атрибутов.

Абсолютный путь доступа

Путь доступа, начинающийся со знака /, за которым следуют один или несколько компонентов, разделенных знаком /. Все пути доступа, начинающиеся с /, обрабатываются от корневого узла, поэтому они всегда возвращают одинаковый результат независимо от текущего контекстного узла. См. также *путь доступа* и *относительный путь доступа*.

Аннотация типа [2.0]

Тип данных, связываемый с каждым узлом элемента и атрибута. Аннотации типов добавляются в узлы элементов и атрибутов при проверке их действительности по схеме. Всем узлам, не прошедшим проверку по схеме, назначается тип данных `xs:untyped`; всем непроверенным атрибутам назначается тип данных `xs:untypedAtomic`.

Арность [2.0]

Количество параметров, передаваемых при вызове функции.

Атомарное значение [2.0]

Значение одного из атомарных типов данных, определяемых XML Schema (или типа данных, производного от одного из этих типов). Примеры атомарных типов данных: `xs:integer`, `xs:dayTimeDuration` и `xs:string`.

Базовый URI-адрес

URI-адрес, связываемый с каждым узлом исходного дерева XPath. В некоторых обстоятельствах базовый URI-адрес используется для разрешения ссылок на другие ресурсы. Если некоторый узел является узлом элемента или узлом инструкции по обработке и находится во внешней сущности, базовым URI-адресом этого узла является базовый URI-адрес внешней сущности. Если узел элемента

или узел инструкции по обработке не находится во внешней сущности, его базовым URI-адресом является базовый URI-адрес документа, в котором он находится. Базовым URI-адресом узла документа является базовый URI-адрес самого документа, а базовым URI-адресом узла атрибута, комментария, пространства имен или текста является базовый URI-адрес родителя этого узла.

Выражение пути

Выражение, выбирающее узлы из дерева. Выражение строится из *компонентов*. Узлы, возвращаемые выражением пути, следуют в *порядке документа* с удалением всех дубликатов.

Действительный документ

Документ XML, удовлетворяющий основным правилам синтаксиса документов XML, а также выполняющий все правила связанного с ним DTD или схемы. За подробным описанием правил документов XML обращайтесь к разделу «Правила документов XML» главы 1; в разделе «DTD (Document Type Definition) и XML Schema» главы 1 рассматриваются определения типов и схемы.

Дерево результата

Дерево, образующее часть итогового вывода преобразования. Однажды созданное содержимое дерева результата становится недоступным для таблицы стилей.

Динамический контекст [2.0]

Часть *контекста*, которая может изменяться в ходе обработки таблицы стилей или выражения XPath. Контекстный объект, размер и позиция контекста являются компонентами динамического контекста. За полной информацией о компонентах контекста обращайтесь к разделу «[2.0] Контекст XPath 2.0» главы 3. См. также *статический контекст [2.0]*.

Дочерний элемент

Непосредственный потомок заданного узла (сравните с *потомок*). К дочерним узлам элемента относятся все узлы комментариев, элементов, инструкций по обработке и текста. Узлы атрибутов и пространств имен не считаются дочерними узлами.

Инструкция по обработке

Часть документа XML, содержащая инструкции для приложений. Пример инструкции по обработке:

```
<?xml-stylesheet href="docbook/html/
   docbook.xsl" type="text/xsl"?>
```

Эта инструкция связывает таблицу стилей XSLT с документом XML. Инструкции по обработке подробно рассматриваются в разделе «DOM (Document Object Model)» главы 1.

Ключ

Аналог индекса в базе данных. Состоит из трех компонентов: *имени*, по которому идентифицируется ключ (атрибут `name` элемента `<xsl:key>`), *узлов*, возвращае-

мых ключом (атрибут use элемента <xsl:key>), и *значений*, используемых для поиска по ключу (атрибут match элемента <xsl:key>).

Например, элемент <key name="language-index" match="defn" use="@language"/> определяет новый ключ с именем language-index. Ключ возвращает все элементы <defn>, у которых атрибут language совпадает с заданным значением атрибута language. За подробной информацией о ключах и их использовании обращайтесь к разделу «Элементы программной логики в XSLT» главы 5.

Код символа [2.0]

Число, представляющее символ Юникода. Например, число x20AC является кодом символа «знак евро» (€).

Кодировка

Набор символов, указанный в объявлении XML для описания некоторого документа. Допустимые значения определены в *http://www.ietf.org/rfc/rfc2278.txt*. Состав поддерживаемых значений зависит от конкретного парсера XML или процессора XSLT.

Компонент

Часть выражения пути, определяющая перемещение между узлами. Компонент оси, тест узла и предикатное выражение являются компонентами в выражении пути.

Компонент пути доступа

Состоит из трех частей: имени оси, теста узла и нуля или более предикатных выражений. Например, выражение XPath preceding-sibling::region/product[@name="Sandpiper"]/text() содержит три компонента пути доступа.

Первый компонент, preceding-sibling::region, состоит из имени оси preceding-sibling и теста узла region. Он выбирает все элементы <region>, являющиеся предшествующими одноуровневыми узлами для контекстного узла. Компонент не содержит предикатного выражения.

Второй компонент, product[@name="Sandpiper"], содержит имя оси child (эта ось используется по умолчанию). В него входят также тест узла product и предикат [@name="Sandpiper"]. Он выбирает все дочерние элементы <product> предыдущего компонента, содержащие атрибут name со значением Sandpiper.

Третий компонент, text(), состоит из имени оси child и теста узла text(). Он выбирает все текстовые дочерние узлы предыдущего компонента. Компонент не содержит предикатного выражения.

Контекст

Структура данных, определяющая способ обработки выражений XPath.

[1.0] В XPath 1.0 контекст состоит из пяти компонентов: контекстного узла, пары положительных целых чисел (позиция и размер контекста), набора привязок переменных, библиотеки функций и набора объявлений пространств имен в области видимости.

[2.0] В XPath 2.0 контекст содержит все компоненты контекста XPath 1.0 и ряд новых компонентов, в том числе типы данных и пространства имен, находящие-

ся в области видимости. За подробным описанием контекста обращайтесь к разделу «Контекст XPath» в приложении B.

Контекстный объект [2.0]

Объект, являющийся отправной точкой для обработки всех выражений XPath. В XPath 2.0, если контекстный объект является узлом, контекстный узел совпадает с контекстным объектом. Если контекстный объект является атомарным значением, контекстный узел не определен.

Контекстный узел

Узел, являющийся отправной точкой для обработки всех выражений XPath. Контекстный узел является аналогом текущего каталога в командной строке; все команды, вводимые в командной строке, обрабатываются в контексте текущего каталога. См. также *текущий узел*.

[2.0] В XPath 2.0 контекстный узел не определен, если контекстный объект является атомарным значением. Если контекстный объект является узлом, контекстный узел совпадает с контекстным объектом.

Корневой узел

Узел XPath, представляющий корень документа XML. Обратите внимание: корневой узел и узел элемента, представляющий элемент документа, – разные узлы. Корневой узел задается выражением XPath /. Дочерними узлами корневого узла являются узел элемента, представляющий элемент документа, а также все комментарии и инструкции по обработке, находящиеся за пределами элемента документа. *[2.0]* В XSLT 2.0 корневой узел, представляющий документ XML, называется узлом документа.

Корректный документ

Документ XML, удовлетворяющий основным правилам синтаксиса XML. За подробным описанием этих правил обращайтесь к разделу «Правила документов XML» главы 1.

Литеральный элемент результата (LRE)

Элемент таблицы стилей XSLT, не принадлежащий пространству имен XSLT и не являющийся элементом расширения. Литеральные элементы просто копируются в дерево результата.

Локальное имя

Неуточненная часть имени элемента или атрибута. Например, в элементе `<xsl:template>` часть `template` является локальным именем.

Набор атрибутов

Именованная группа атрибутов. Разработчик может создать набор атрибутов (при помощи элемента `<xsl:attribute-set>`), а затем сослаться на него в другом месте. За дополнительной информацией обращайтесь к описанию элемента `<xsl:attribute-set>` в приложении A.

Неразобранная сущность

Ресурс в документе XML, содержимое которого может быть (или не быть) текстом, причем текст не обязан храниться в формате XML. С каждой неразобранной сущностью связывается запись в формате XML. За дополнительной информацией о неразобранных сущностях обращайтесь к описанию функции `unparsed-entity-uri()` в приложении C.

Неразобранный текст [2.0]

Текст, возвращаемый функцией `unparsed-text()`. Как подсказывает имя функции, текст не разбирается, поэтому он может быть или не быть действительным кодом XML, корректным кодом XML или какой-то иной разновидностью разметки.

Объект [2.0]

Значение, которое является либо атомарным значением, либо узлом.

Объявление XML

Часть документа XML, определяющая используемую версию XML и кодировку документа. Хотя объявление XML выглядит как инструкция по обработке, оно ею не является. По этой причине к объявлению XML невозможно обратиться из таблицы стилей XSLT или выражения XPath.

Объявление пространства имен

Часть документа XML, определяющая пространство имен для конкретной части документа. Объявления пространств имен оформляются в виде атрибутов XML. Объявление пространства имен, начинающееся с `xmlns=`, определяет пространство имен по умолчанию, а объявление, начинающееся с `xmlns:prefix`, определяет пространство имен с префиксом.

Одноуровневые узлы

Два узла, имеющие общего родителя (sibling). В XPath определяются оси `preceding-sibling` и `following-sibling`.

Оси

Отношения между контекстным узлом и другими узлами документа. XPath определяет 13 различных осей; за полной информацией обращайтесь к разделу «Оси» главы 3.

Откат

Действия, выполняемые для корректной обработки отсутствия элемента или функции расширения. Как правило, обработка выполняется по результатам вызова функции `element-available()` или `function-available()`. Если функция возвращает `false`, таблица стилей корректно реагирует на недоступность указанной функции. В XSLT определяется также элемент `<xsl:fallback>`, который может использоваться при недоступности элемента расширения.

Относительный путь доступа

Путь доступа, состоящий из одного или нескольких компонентов, разделенных знаком /. См. также *путь доступа* и *абсолютный путь доступа*.

Параметр

Механизм XSLT, связывающий имена со значениями; определяется элементом `<xsl:param>`. Параметры отличаются от переменных тем, что значение, заданное в определении параметра, является значением по умолчанию. При вызове шаблона или таблицы стилей, содержащей параметр, значение по умолчанию можно переопределить. Но, как и в случае с переменными, единожды заданное значение параметра изменить уже нельзя.

Переменная

Механизм XSLT, связывающий имя со значением; определяется элементом `<xsl:variable>`. Переменные отличаются от параметров тем, что параметры могут иметь значения по умолчанию. Еще одно важное различие между переменными XSLT и переменными других языков программирования заключается в том, что после связывания переменной XSLT со значением последнее не может быть изменено. За подробным описанием элемента `<xsl:variable>` и его использования обращайтесь к разделу «Переменные» главы 5.

Позиция контекста

[1.0] Положительное целое число, обозначающее позицию текущего узла. Позиция контекста всегда меньше размера контекста либо равна ему.

[2.0] Положительное целое число, обозначающее позицию текущего объекта в последовательности объектов, обрабатываемой в данный момент.

Полное имя (QName)

Имя элемента или атрибута, уточненное префиксом пространства имен. Полное имя записывается в формате `prefix:local-name`, где и необязательный префикс `prefix`, и локальное имя `local-name` представляют собой имена, не разделенные двоеточием (NCName). Например, в таблице стилей XSLT `<xsl:template>` – полное имя с префиксом, а `<template>` – нет. Имена в таблицах стилей XSLT (имена переменных, имена шаблонов, имена режимов и т. д.) являются полными именами независимо от того, имеется у них префикс или нет. Обратите внимание: если пространство имен по умолчанию является пустым (`xmlns=""`), полное имя без префикса не имеет пространства имен.

Порядок документа

Порядок следования узлов в исходном документе XML.

Последовательность

Упорядоченная совокупность *объектов*.

Потомок

Дочерний узел заданного узла, дочерние узлы дочернего узла и т. д. Сравните с *дочерний узел*.

Предикатное выражение

Выражение XPath в квадратных скобках ([ ]). Предикатные выражения фильтруют *[1.0]* набор узлов или *[2.0]* последовательность, выбирая из них только узлы, соответствующие выражению в квадратных скобках. За дополнительной информацией обращайтесь к разделу «Предикаты» главы 3.

Предок

Узел, находящийся выше заданного узла. К предкам относятся родитель узла, родитель родителя и т. д. В XPath определяется также ось `ancestor`, которая включает родителя узла, родителя его родителя и т. д., но не сам узел. Сравните с *родитель*.

Преобразование типа [2.0]

Преобразование значения от одного атомарного типа к другому.

Префикс

Сокращение от *префикс пространства имен*.

Префикс пространства имен

Часть полного имени, связывающая элемент или атрибут с URI-адресом пространства имен.

Примитивный тип [2.0]

В XML Schema определяются 19 примитивных типов. В модели данных XPath 2.0 все они определяются как подтипы `xs:anyAtomicType`. (19 примитивных типов: `xs:boolean`, `xs:string`, `xs:decimal`, `xs:double`, `xs:float`, `xs:QName`, `xs:anyURI`, `xs:hexBinary`, `xs:base64Binary`, `xs:date`, `xs:time`, `xs:dateTime`, `xs:gYear`, `xs:gYearMonth`, `xs:gMonth`, `xs:gMonthDay`, `xs:gDay`, `xs:duration` и `xs:NOTATION`.) XPath добавляет также `xs:untypedAtomic` — тип данных любого значения, которое не было проверено по схеме или преобразовано к примитивному типу.

Проверка действительности

Процесс сравнения узла со схемой. Если узел действителен, то ему, а также всем его дочерним узлам и атрибутам назначается тип данных, основанный на определениях узлов в схеме.

Пропуск

Один из четырех символов: пробел (`&#x20`), табуляция (`&#x09`), возврат курсора (`&#x0D`) или перевод строки (`&#x0A`).

Пространство имен

Совокупность имен элементов и атрибутов, связанная с URI-адресом.

Пустая последовательность

Последовательность, содержащая нуль объектов.

Путь доступа

Выражение XPath, выбирающее набор узлов относительно контекстного узла. См. также *абсолютный путь доступа* и *относительный путь доступа*.

Размер контекста

[1.0] Количество узлов в текущем списке узлов.

[2.0] Количество объектов в последовательности объектов, обрабатываемой в данный момент.

Расширенное имя

Полное имя элемента или атрибута, включающее локальное имя и URI-адрес пространства имен (который может быть пустым).

Регулярное выражение [2.0]

Синтаксис анализа строк. Регулярные выражения используются функциями XPath 2.0 `matches()`, `replace()` и `tokenize()`, а также инструкцией XSLT 2.0 `<xsl:analyze-string>`.

Режим

Механизм XSLT, обеспечивающий многократную обработку элемента по разным шаблонам с получением различных результатов. За подробным описанием режимов обращайтесь к описанию элемента `<xsl:apply-templates>` в приложении А.

Родитель

Узел, находящийся непосредственно над заданным узлом. Родитель является ближайшим предком узла. XPath определяет также ось `parent`, содержащую родителя узла. Все узлы исходного документа, за исключением корневого узла, имеют родителя. Сравните с *предок*.

Составной тип [2.0]

Любой тип данных, определяемый в XML Schema, который может иметь дочерние узлы или атрибуты.

Статическая ошибка

Ошибка, которая может быть обнаружена до обработки исходного документа XML.

Статический контекст [2.0]

Часть *контекста*, известная до начала обработки исходного документа XML. За полным описанием всех составляющих контекста обращайтесь к разделу «[2.0] Контекст XPath 2.0» главы 3. См. также *динамический контекст [2.0]*.

Схема упорядочения [2.0]

Набор правил, определяющих способ сравнения строковых значений. В XSLT 2.0 и XPath 2.0 процессоры могут поддерживать пользовательские схемы упорядочения, что позволяет выполнять сравнения по правилам разных языков. Например, слово «улица» в немецком языке может записываться в двух вариантах:

Strasse и *Straße*; в пользовательской схеме упорядочения эти две строки могут считаться равными.

Таблица стилей

Документ XML, написанный в номенклатуре XSLT и описывающий правила преобразования другого документа XML.

Текстовый узел

Группа символов из документа XML. Текстовые узлы составляют одну из семи разновидностей узлов, определяемых в XPath. Спецификация XPath указывает, что один текстовый узел должен содержать максимально возможный объем текста. Другими словами, текстовый узел никогда не имеет предшествующих или следующих одноуровневых узлов, которые также являются текстовыми узлами.

Текущий список узлов

Список узлов, выбранных атрибутом `select` элемента `<xsl:apply-templates>` или `<xsl:for-each>`, обрабатываемого в данный момент. По умолчанию узлы в списке следуют в порядке документа, но их порядок может быть изменен одним или несколькими элементами `<xsl:sort>`.

[2.0] В XPath 2.0 текущий список узлов называется «последовательностью объектов, обрабатываемой в данный момент».

Текущий узел

[1.0] Узел, обрабатываемый в данный момент. Определяется атрибутом `select` элемента `<xsl:apply-templates>` или `<xsl:for-each>`. За исключением предикатных выражений, текущий узел совпадает с контекстным узлом.

[2.0] Объект, обрабатываемый в данный момент.

Тест узлов

Выражение XPath, выбирающее некоторые узлы. Выражения `child::*`, `para` и `@id` выбирают все дочерние узлы, все дочерние узлы `<para>` и все атрибуты с именем `id` соответственно. Четыре теста узлов — `text()`, `comment()`, `node()` и `processing-instruction()` — выглядят как функции, хотя формально ими не являются. (*[2.0]* В XSLT 2.0 поддерживается также тест узлов `document-node()`.) За полным списком тестов узлов XPath обращайтесь к разделу «Тесты узлов XPath» приложения B.

Туннельный параметр [2.0]

Туннельный параметр автоматически передается всем шаблонам, вызываемым из текущего шаблона. Далее туннельный параметр рекурсивно передается всем последующим шаблонам в цепочке вызова.

Узел атрибута

Тип узла XPath, представляющего атрибут документа XML. Атрибуты отличаются от других узлов тем, что узел атрибута не считается дочерним узлом того узла элемента, в котором он содержится. Несмотря на это обстоятельство, узел элемента считается родителем узла атрибута.

Узел документа

В дереве, созданном на базе корректного документа XML, корневым узлом является узел документа с ровно одним дочерним узлом элемента. Этот узел элемента представляет элемент документа и не имеет дочерних текстовых узлов. Парсер XML создает узлы документов только на базе корректных документов XML. Тем не менее таблица стилей XSLT может создать узел документа, который не является корректным. Пример приведен в описании элемента `<xsl:document>`.

Узел инструкции по обработке

Тип узла XPath, представляющего инструкцию по обработке из документа XML.

Узел комментария

Узел, представляющий комментарий из исходного документа XML. Одна из семи разновидностей узлов, определяемых в XPath.

Узел пространства имен

Тип узла XPath, соответствующий объявлению пространства имен в документе XML.

Узел текста

Группа символов из документа XML. Узлы текста являются одной из семи разновидностей узлов, определяемых в XPath. Согласно спецификации XPath один узел текста должен содержать максимально возможный объем текста; следовательно, следующий или предыдущий одноуровневый узел текстовым быть не может.

Узел элемента

Узел XPath, представляющий элемент исходного документа XML.

Упорядочение по кодам символов [2.0]

Схема упорядочения, которая сравнивает строки по кодам символов. Используется по умолчанию всеми процессорами XSLT.

Фрагмент дерева результата [1.0]

Фрагмент дерева результата, который может связываться с переменной в XSLT 1.0. За подробным описанием фрагментов дерева результата обращайтесь к разделу «Типы данных XPath 1.0» в приложении B. *[2.0]* В XSLT 2.0 понятие фрагмента дерева результата заменяется последовательностью.

Функция-конструктор [2.0]

Функция, используемая для создания атомарного значения определенного типа. Имя функции совпадает с именем типа данных; функция `xs:integer()` является конструктором для типа данных `xs:integer`.

Функция расширения

Функция, префикс пространства имен которой ссылается на расширение. Спецификация XSLT определяет способ идентификации функций расширения в табли-

цах стилей, но не определяет способ их реализации. Функции расширения реализуются в виде блоков кода, ссылки на которые включаются в таблицы стилей; каждый процессор XSLT сам определяет, как активизировать этот код для выполнения преобразования в функции расширения. За подробным описанием элементов и функций расширения обращайтесь к главе 8.

Шаблон

1 (template). Правило таблицы стилей XSLT, определяющее способ преобразования части документа XML. Шаблоны определяются элементом `<xsl:template>`.

2 (pattern). Условие, при котором идентифицируется (или не идентифицируется) совпадение для данного узла. Синтаксисом шаблона является подмножество синтаксиса выражений XPath. Шаблоны используются в элементах XSLT `<xsl:template>`, `<xsl:key>`, `<xsl:number>` и `<xsl:for-each-group>`.

Шаблон значения атрибута

Выражение, содержащее выражение XPath в фигурных скобках ({}). Выражение XPath обрабатывается во время выполнения, а полученное значение подставляется на место выражения. Примеры шаблона значения атрибута приведены в описании элемента `<xsl:attribute>` в приложении А; в главе 3 содержится полное описание шаблонов значений атрибутов.

Экранирование вывода

Процесс замены зарезервированных символов (таких как <, > и &) соответствующими символьными сущностями (такими как `<`, `>` и `&`).

Элемент верхнего уровня

Элемент, родителем которого является элемент `<xsl:stylesheet>`.

Элемент документа

Элемент исходного документа XML, содержащий весь документ XML. Узел, представляющий элемент документа, является дочерним для корневого узла; корневой узел и узел элемента, представляющий элемент документа, – разные узлы.

Элемент расширения

Элемент таблицы стилей XSLT, префикс пространства имен которого содержит ссылку на расширение. Спецификация XSLT определяет способ идентификации элементов расширения в таблицах стилей, но не определяет способ их реализации. Элементы расширения реализуются в виде блоков кода, ссылки на которые включаются в таблицы стилей; каждый процессор XSLT сам определяет, как активизировать этот код для выполнения преобразования в элементе расширения. Подробное описание элементов и функций расширения см. в главе 8.

Алфавитный указатель

Специальные символы

!= (не равно), оператор, 584
#all, значение атрибута mode, 190
#current, значение атрибута mode, 190
#default, значение атрибута mode, 190
$ (доллар), метасимвол, 911
( ) (круглые скобки), 584
((левая круглая скобка), метасимвол, 911
) (правая круглая скобка), метасимвол, 911
\(экранированный символ, 912
\) экранированный символ, 912
* (звездочка)
  as, атрибут, 75
  квантификатор, 584, 906
  метасимвол, 911
  оператор умножения, 584
* экранированный символ, 912
@*, метасимвол, 85
+ (плюс)
  квантификатор, 584, 906
  метасимвол, 911
  оператор сложения, 584
  унарный оператор, 584
\+ экранированный символ, 912
, (запятая), оператор последовательности, 585
. (точка), метасимвол, 910
\. экранированный символ, 912
- (дефис), метасимвол, 911
- (минус)
  оператор вычитания, 585
  унарный оператор, 585
\- экранированный символ, 912
/ (косая черта)
  в путях доступа, 585
// (двойная косая черта)
  в путях доступа, 585
  оператор, 86
\ (обратная косая черта), метасимвол, 911
\\ экранированный символ, 912
? (вопросительный знак)
  квантификатор, 586, 906
  метасимвол, 911
\? экранированный символ, 912
[(квадратная скобка), метасимвол, 911
[ ] (квадратные скобки), оператор, 586
\[экранированный символ, 912
\] экранированный символ, 912
^, метасимвол, 911
{} (фигурные скобки), в регулярных выражениях, 419
\{ экранированный символ, 912
\} экранированный символ, 912
| (вертикальная черта), оператор, 124, 586, 911
\| экранированный символ, 912
> (больше), оператор, 585
>= (больше или равно), оператор, 586
< (меньше), оператор, 176, 585
<= (меньше или равно), оператор, 176, 585
= (равно), оператор, 585
<< (узел-до), оператор, 129, 585
>> (узел-после), оператор, 128, 586

A

abs(), функция, 596
adjust-date-to-timezone(), функция, 598
adjust-dateTime-to-timezone(), функция, 600
adjust-time-to-timezone(), функция, 602
Altova XSLT
  глобальные параметры, 187

команды в примере Hello World, 44
установка, 44
ancestor, ось, 88, 577
ancestor-or-self, ось, 88, 134, 577
ancestor::table[], предикат, 90
and, оператор, 586
as, атрибут, 75, 471, 555
attribute(), тест узлов, 74, 576
attribute, ось, 88, 577
avg(), функция, 604

B

base-uri(), функция, 607
bgcolor, атрибут, 184
boolean(), функция, 610
BSF (Bean Scripting Framework)
SVG, пример, 365
byte-order-mark, атрибут, 515

C

C, символьные группы, 915
\C, служебная комбинация, 913
\c, служебная комбинация, 913
case-order, атрибут, 246, 542
cast as, оператор, 121, 586
castable as, оператор, 122, 421, 586
Cc, символьная группа, 915
CDATA, секция, 933
CDATA, узлы, текст, 72
cdata-section-elements, атрибут, 514
ceiling(), функция, 614
Cf, символьная группа, 915
character, атрибут, 519
child, ось, 87, 577
Cn, символьная группа, 915
Co, символьная группа, 915
codepoint-equal(), функция, 616
codepoints-to-string(), функция, 617
collation, атрибут, 226, 462, 483, 543
collection(), функция, 307, 618
comment(), тест узлов, 73, 84, 576
compare(), функция, 328, 620
concat(), функция, 202, 623
contains(), функция, 625
copy-namespaces, атрибут, 438
count, атрибут, 150, 499
count(), предикат, 90
count(), функция, 628
CSS
общие сведения, 20
сравнение с XSLT, 140
CSS, стили, в документах HTML, 436
current(), функция, 631
current-date(), функция, 634
current-dateTime(), функция, 635
current-group(), функция, 263, 636
current-grouping-key(), функция, 263, 638
current-time(), функция, 640

D

d, атрибут, 353
D, код форматирования, 161
\D, служебная комбинация, 913
\d, служебная комбинация, 913
D1o, код форматирования, 161
data(), функция, 642
data-type, атрибут, 243, 542
dateTime(), функция, 645
day-from-date(), функция, 646
day-from-dateTime(), функция, 648
days-from-duration(), функция, 649
decimal-separator, атрибут, 445
deep-equal(), функция, 650
default-collation(), функция, 655
default-collation, атрибут, 401, 551
default-validation, атрибут, 551
descendant, ось, 88, 577
descendant-or-self, ось, 88
digit, атрибут, 446
disable-output-escaping, атрибут, 556
distinct-values(), функция, 251, 656
div, оператор, 586
doc(), функция, 305, 660
doc-available(), функция, 306, 662
DocBook, документы
group-starting-with, 274
<xsl:for-each-group>, 466
преобразование в HTML, 195
doctype-public, атрибут, 514, 535
doctype-system, атрибут, 514, 535
document(), функция, 279, 663
document-node(), тест узлов, 75, 576
document-uri(), функция, 665
DOM
деревья, 32
общие сведения, 40
объединение данных, 280
программный интерфейс, 32
элементы расширения, 384

DTD
правила, 28
сравнение с XML Schema, 28
Dwo, код форматирования, 161
dwo, код форматирования, 161

E

EcmaScript, код, 341
element(), тест узлов, 74, 576
element-available(), функция, 318, 666
elements, атрибут, 530
else, ключевое слово, 115
empty(), функция, 670
encode-for-uri(), функция, 671
encoding, атрибут, 514
ends-with(), функция, 673
eq (равно), оператор, 586
error(), функция, 675
escape-html-uri(), функция, 678
escape-uri-attributes, атрибут, 515, 535
every, оператор, 587
exactly-one(), функция, 679
except, оператор, 125, 587
exclude-result-prefixes, атрибут, 401, 551
EXIF, теги, 398
exists(), функция, 681
EXSLT, библиотека, 369
extension-element-prefixes, атрибут, 319, 401, 551

F

false(), функция, 682
flags, атрибут, 402
floor(), функция, 683
FNn, код форматирования, 161
following, ось, 89, 578
following-sibling, ось, 88, 578
for, оператор, 115, 587
format, атрибут, 148, 500, 534
format-date(), функция, 159, 300, 684
format-dateTime(), функция, 159, 687
format-number(), функция, 156, 287, 690, 922
format-time(), функция, 159, 693
from, атрибут, 500
function-available(), функция, 332, 350, 371, 695

G

gDay, тип данных, 94
ge (больше или равно), оператор, 587
generate-id(), функция, 235, 698
gMonth, тип данных, 94
group-adjacent, атрибут, 266, 462
group-adjacent, стиль группировки, 262
group-by, атрибут, 262, 462
group-by, стиль группировки, 262
group-ending-with, атрибут, 275, 462
group-ending-with, стиль группировки, 262, 275
group-starting-with, атрибут, 273, 462
group-starting-with, стиль группировки, 262, 273
grouping-separator, атрибут, 445, 501
grouping-size, атрибут, 501

H

h01, код форматирования, 161
hours-from-dateTime(), функция, 701
hours-from-duration(), функция, 702
hours-from-time(), функция, 703
href, атрибут, 476, 481, 534
HTML
<code>, элемент, 495
group-adjacent, атрибут, 266
group-starting-with, атрибут, 274
<xsl:key>, 259
генерирование на базе XML, 142
общие сведения, 23
преобразование документов DocBook, 195
пропуски, 142
ссылки, 220
стили CSS, 436
сценарный код, 340
таблицы, 378
Hwo, код форматирования, 161

I

\I, служебная комбинация, 913
\i, служебная комбинация, 913
i, флаг, 402, 908
id(), функция, 215, 216, 705
id, атрибут, 551
ID, тип данных
ограничения, 224
определение, 933

idiv, оператор, 587
idref(), функция, 216, 709
IDREF, тип данных, 212, 933
IDREFS, тип данных, 212, 933
if, оператор, 114, 272, 588
implicit-timezone(), функция, 713
in-scope-prefixes(), функция, 714
include-content-type, атрибут, 515
indent, атрибут, 514
index-of(), функция, 298, 716
infinity, атрибут, 445
inherit-namespaces, атрибут, 439, 453
input-type-annotations, атрибут, 551
insert-before(), функция, 718
instance of, оператор, 120, 588
intersect, оператор, 126, 588
iri-to-uri(), функция, 722
is, оператор, 127, 588

J

Jacl, интерпретатор, 368
Java, программы
  Hello World, 61
    команды для примера, 62
  глобальные параметры, 188
JavaScript, ядро, 367
JDBC, источник данных, 374
JRuby, язык, 366
Jython, 365

K

key(), функция, 225, 258, 723

L

L, символьная группа, 913
lang(), функция, 726
lang, атрибут, 246, 366, 501, 542
last(), функция, 727
le, оператор, 588
letter-value, атрибут, 501
level, атрибут, 499
line[], предикат, 90
Ll, символьная группа, 913
Lm, символьная группа, 913
Lo, символьная группа, 913
local-name(), функция, 729
local-name-from-QName(), функция, 731
lower-case(), функция, 732
Lt, символьная группа, 913
Lu, символьная группа, 913

M

M, символьная группа, 914
m, флаг, 402, 908
m01, код форматирования, 161
m1o, код форматирования, 161
main, метод, 50
match, атрибут, 83, 226, 483, 554
matches(), функция, 734
max(), функция, 737
Mc, символьная группа, 914
Me, символьная группа, 914
media-type, атрибут, 514
method, атрибут, 513, 535
min(), функция, 741
minus-sign, атрибут, 445
minutes-from-dateTime(), функция, 745
minutes-from-duration(), функция, 746
minutes-from-time(), функция, 748
Mn, символьная группа, 914
MNn, код форматирования, 161
MNn,3-3 код форматирования, 161
mod, оператор, 105, 152, 588
mode, атрибут, 183, 411, 554
month-from-date(), функция, 749
month-from-dateTime(), функция, 750
month-name(), функция, 370
months-from-duration(), функция, 752

N

N, символьная группа, 914
\n, служебная комбинация, 912
name, атрибут, 219, 415, 452, 483, 532, 569
name(), функция, 753
namespace, атрибут, 416, 478
namespace, ось, 89, 578
namespace-uri(), функция, 756
namespace-uri-for-prefix(), функция, 759
namespace-uri-from-QName(), функция, 761
NaN, атрибут, 445
NCName(), тест узлов, 74
Nd, символьная группа, 914
ne, оператор, 588
.NET Framework
  глобальные параметры, 187
  команды для примера Hello World, 44
  общие сведения, 11

установка процессора, 43
nilled(), функция, 765
Nl, символьная группа, 914
No, символьная группа, 914
node(), тест узлов, 73, 85, 576
node-name(), функция, 767
normalization-form, атрибут, 515, 535
normalize-space(), функция, 172, 771
normalize-unicode(), функция, 773
not(), функция, 108, 774
number(), функция, 776

O

omit-xml-declaration, атрибут, 514, 536
one-or-more(), функция, 779
or, оператор, 589
order, атрибут, 246, 542
ordinal, атрибут, 156, 501
output-version, атрибут, 536
override, атрибут, 471

P

P, код форматирования, 161
P, символьная группа, 914
\P, служебная комбинация, 913
\p, служебная комбинация, 913
parent, ось, 87, 577
pattern-separator, атрибут, 446
Pc, символьная группа, 914
Pd, символьная группа, 914
PDF-файлы, пример Hello World, 59
Pe, символьная группа, 914
per-mille, атрибут, 445
percent, атрибут, 445
Pf, символьная группа, 914
Pi, символьная группа, 914
Po, символьная группа, 914
position(), функция, 277, 781
preceding, ось, 89, 578
preceding-sibling, ось, 88, 578
prefix-from-QName(), функция, 784
priority, атрибут, 555
processing-instruction(), тест узлов, 73, 84, 576
Ps, символьная группа, 914

Q

QName(), функция, 786

R

\r, служебная комбинация, 912
regex, атрибут, 402
regex-group(), функция, 788, 904
RelaxNG, язык схем, 39
remove(), функция, 790
replace(), функция, 205, 795, 906
required, атрибут, 522
resolve-QName(), функция, 799
resolve-uri(), функция, 802
result tree fragment, тип данных, 580
result-prefix, атрибут, 492
reverse(), функция, 803
RFC 2396 и RFC2732, xs:anyURI, 95
root(), функция, 805
rotate, оператор, 353
round(), функция, 808
round-half-to-even(), функция, 810

S

S, символьные группы, 915
\S, служебная комбинация, 912
\s, служебная комбинация, 912
s, флаг, 402, 907
SAX
  общие сведения, 40
  программный интерфейс, 32
Saxon, процессор
  базы данных, 372
  глобальные параметры, 187
  команды примера Hello World, 42
  неопределенные параметры, 192
  общие сведения, 11
  передача глобальных параметров, 524
  пользовательские схемы упорядочения, 323
  упорядочение при сортировке, 324
  установка, 42
Sc, символьная группа, 915
schema-element(), тест узлов, 74, 576
schema-location, атрибут, 478
Schematron, 39
seconds-from-dateTime(), функция, 812
seconds-from-duration(), функция, 813
seconds-from-time(), функция, 814
select, атрибут, 133, 250, 401, 442, 487, 501, 532
self, ось, 87, 577
separator, атрибут, 144, 416, 559

SGML, 23
Sk, символьная группа, 915
Sm, символьная группа, 915
some, оператор, 115, 589
SQL, команды, на базе документов XML, 201
SQL, поддержка процессором Saxon, 372
sql:close(), функция, 374
<sql:connect>, элемент, 374
<sql:connection>, элемент, 373
sql:disableStreamingMode(), функция, 376
sql:new(), функция, 378
sql:query(), функция, 374
<sql:query>, элемент, 373
stable, атрибут, 543
standalone, атрибут, 514, 536
starts-with(), функция, 815
static-base-uri(), функция, 818
string(), функция, 820
string, атрибут, 520
string-join(), функция, 824
string-length(), функция, 826
string-to-codepoints(), функция, 828
stylesheet-prefix, атрибут, 492
subsequence(), функция, 310, 829
substring(), функция, 832
substring-after(), функция, 202, 834
substring-before(), функция, 202, 837
sum(), функция, 284, 839
SVG
  круговые диаграммы, 370
  общие сведения, 41
system-property(), функция, 843

T

\t, служебная комбинация, 912
<td>, элемент, 418
terminate, атрибут, 487
test, атрибут, 176, 474, 566
text(), тест узлов, 73, 83, 576
timezone-from-date(), функция, 848
timezone-from-dateTime(), функция, 849
timezone-from-time(), функция, 850
tokenize(), функция, 309, 851
toRadians(), функция, 371
trace(), функция, 854
transform, атрибут, 353
translate(), функция, 857
translate, оператор, 353
treat as, оператор, 122, 589
true(), функция, 859
tunnel, атрибут, 522, 569
type, атрибут, 439, 534
type-available(), функция, 860

U

undeclare-prefixes, атрибут, 515, 536
unordered(), функция, 862
unparsed-entity-public-id(), функция, 864
unparsed-entity-uri(), функция, 865
unparsed-text(), функция, 309, 866
unparsed-text-available(), функция, 311, 869
upper-case(), функция, 870
URI-адреса пространств имен, 934
use, атрибут, 483
use-attribute-sets, атрибут, 421, 453
use-character-maps, атрибут, 429, 536
use-when, атрибут, 401
UTC (Universal Coordinated Time), 94

V

validation, атрибут, 439, 534
value, атрибут, 500
version, атрибут, 400, 513, 550
VRML, пример Hello World, 63

W

\W, служебная комбинация, 913
\w, служебная комбинация, 913
width, атрибут, 184

X

x, флаг, 402, 908
Xalan, процессор XSLT
  <redirect:write>, элемент, 318
  базы данных, 376
  глобальные параметры, 187
  команды для примера Hello World, 42, 62
  неопределенные параметры, 191
  общие сведения, 12
  передача глобальных параметров, 524
  установка, 41
Xalan, расширения Java, 380
XDM (XPath 2.0 Data Model), 37
Xlink (XML linking language)
  общие сведения, 41

XML
  история, 19
  правила, 24
  программные интерфейсы, 32
  стандарты, 38
  стандарты XSLT, 36
XML Schema, 878
  XSLT 2.0, 453, 930
  встроенные типы данных, 68
  поддержка, 30
  преимущества перед DTD, 29
  стандарты, 38
  таблицы стилей, 896
  типы данных, 581, 882
  элементы и атрибуты, 878
xml:base, атрибут, 288
xml:lang, атрибут, 72
xml:space, атрибут, 72, 172, 530, 548
xmlns:xsl, атрибут, 550
XPath, 67, 574
  ключевые слова, 584
  комментарии в выражениях, 129
  контекст, 578
  логические операторы, 107
  минимальные квантификаторы, 906
  модель данных, 68
  операторы, 96, 584
  оси, 577
  приоритет операторов, 589
  процессоры XSLT, 131
  стандарты, 37
  тесты узлов, 575
  типы данных, 91, 580
  типы узлов, 574
  шаблоны значений атрибутов, 90
XPath 2.0
  collection(), функция, 307
  compare(), функция, 328
  distinct-values(), функция, 251
  every и some, операторы, 115
  index-of(), функция, 298
  replace(), функция, 205
  атомарные значения, 109
  вычитание, 100
  деление, 103
  комментарии в выражениях, 129
  конструкторы, 119
  контекст, 79, 936
  операторы, 114
  операторы типов данных, 119
  операторы узлов, 127
  последовательности и атомарные значения, 75
  приоритет операторов, 590
  сложение, 99
  стандарты, 37
  тесты узлов, 74
  типы данных, 92, 581
  умножение, 102
  язык регулярных выражений, 908
xpath-default-namespace, атрибут, 401, 551
Xpointer (XML pointer language), 41
XQuery 1.0
  compare(), функция, 328
  index-of(), функция, 298
  приоритет операторов, 590
  стандарты, 37
  типы данных, 95
xs:anyAtomicType, тип атомарного значения, 581
xs:anyAtomicType, тип данных, 96
xs:anySimpleType, тип данных, 95
xs:anyType, тип данных, 95
xs:anyURI, тип атомарного значения, 581
xs:anyURI, тип данных, 95
xs:base64Binary, тип атомарного значения, 581
xs:base64Binary, тип данных, 95
xs:boolean, тип атомарного значения, 581
xs:boolean, тип данных, 93
xs:date, тип атомарного значения, 581
xs:date, тип данных, 94
xs:dateTime, тип атомарного значения, 581
xs:dateTime, тип данных, 94
xs:dayTimeDuration, тип атомарного значения, 582
xs:dayTimeDuration, тип данных, 95
xs:decimal, тип атомарного значения, 582
xs:decimal, тип данных, 93
xs:double, тип атомарного значения, 582
xs:double, тип данных, 93
xs:duration, тип атомарного значения, 582
xs:duration, тип данных, 94
xs:float, тип атомарного значения, 582
xs:float, тип данных, 93
xs:gDay, тип атомарного значения, 582
xs:gMonth, тип атомарного значения, 582

xs:gMonthDay, тип атомарного значения, 583
xs:gMonthDay, тип данных, 94
xs:gYear, тип атомарного значения, 583
xs:gYearMonth, тип атомарного значения, 583
xs:gYearMonth, тип данных, 94
xs:hexBinary, тип атомарного значения, 583
xs:hexBinary, тип данных, 94
xs:integer, тип данных, 93
xs:NOTATION, тип атомарного значения, 583
xs:QName, тип атомарного значения, 583
xs:QName, тип данных, 95
xs:string, тип атомарного значения, 583
xs:string, тип данных, 93
xs:time, тип атомарного значения, 583
xs:time, тип данных, 94
xs:untyped, тип атомарного значения, 583
xs:untyped, тип данных, 96
xs:untypedAtomic, тип атомарного значения, 583
xs:untypedAtomic, тип данных, 96
xs:yearMonthDuration, тип атомарного значения, 583
xs:yearMonthDuration, тип данных, 95
XSL, стандарты, 39
xsl, префикс пространства имен, 31
<xsl:analyze-string>, элемент, 401
<xsl:apply-imports>, элемент, 407
<xsl:apply-templates>, элемент, 181, 246, 410
<xsl:attribute-set>, элемент, 421
<xsl:attribute>, элемент, 415
<xsl:call-template>, элемент, 182, 425, 928
<xsl:character-map>, элемент, 429
<xsl:choose>, элемент, 114, 178, 299, 433
<xsl:comment>, элемент, 435
<xsl:copy-of>, элемент, 161
<xsl:copy>, элемент, 161, 438
<xsl:decimal-format>, элемент, 156, 445, 922
<xsl:document>, элемент, 448
<xsl:element>, элемент, 452
<xsl:fallback>, элемент, 315, 458
<xsl:for-each-group>, элемент, 246
<xsl:for-each>, элемент, 180, 240, 262, 460
<xsl:function>, элемент, 296, 470
<xsl:if>, элемент, 176, 299, 474
<xsl:import-schema>, элемент, 478, 896
<xsl:import>, элемент, 54, 475
<xsl:include>, элемент, 54
<xsl:key>, элемент, 55, 225, 258, 483
<xsl:matching-substring>, элемент, 486
<xsl:message>, элемент, 487
<xsl:namespace>, элемент, 490
<xsl:namespace-alias>, элемент, 492
<xsl:next-match>, элемент, 495
<xsl:non-matching-substring>, элемент, 498
<xsl:number>, элемент, 147, 499, 922
<xsl:otherwise>, элемент, 180, 299, 510
<xsl:output>, элемент, 52, 513
<xsl:output-character>, элемент, 519
<xsl:param>, элемент, 55, 183, 521
<xsl:perform-sort>, элемент, 246, 526
<xsl:preserve-space>, элемент, 55, 171, 530
<xsl:processing-instruction>, элемент, 532
<xsl:result-document>, элемент, 315, 533
<xsl:sequence>, элемент, 76, 77, 249, 539
<xsl:sort>, элемент, 237, 289, 542
<xsl:strip-space>, элемент, 55, 171, 548
<xsl:stylesheet>, элемент, 291, 550
<xsl:template>, элемент, 49, 182, 554
<xsl:text>, элемент, 141
<xsl:transform>, элемент, 558
<xsl:value-of>, элемент, 141
<xsl:variable>, элемент, 55, 193, 255, 561
<xsl:when>, элемент, 179, 299, 566
<xsl:with-param>, элемент, 183, 568
XslCompiledTransform, объект, 336
XSLT
  история, 19
  методы вывода, 513
  неопределенные параметры, 190
  передача параметров, 190
  стандарты, 36
  условные выражения, 113
XSLT 2.0
  collation, атрибут, 226
  data-type, атрибут, 243
  doc() и doc-available(), функции, 306
  elements, атрибут, 530
  EXSLT, функции, 369
  idref(), функция, 216
  priority, атрибут, 555
  self, ось, 88

separator, атрибут, 416
to, оператор, 180
unparsed-text() и unparsed-text-available(), функции, 309
xpath-default-namespace, атрибут, 551
<xsl:value-of>, элемент, 143, 558
выражения, 232
вычитание, 100
группировка, 262
деление на нуль, 448
методы вывода, 513
новые возможности, 543
объединение документов XML, 295
параметры, 190
поддержка броузерами, 13
пользовательские схемы упорядочения, 323
последовательности, 115
правила передачи параметров, 427
преобразование к логическим значениям, 176
проверка типов, 930
процессоры, 131
сложение, 97
создание функций, 316
стандарты, 36
типы данных, 32
требования, 22
умножение, 101
форматирование, функции, 159
XSLT, процессоры, 131
EXSLT, 369
установка, 41
XsltArgumentList, объект, 189

Y

year-from-date(), функция, 872
year-from-dateTime(), функция, 873
years-from-duration(), функция, 874

Z

Z, символьные группы, 914
zero-digit, атрибут, 445
zero-or-one(), функция, 876
Zl, символьная группа, 914
Zp, символьная группа, 914
Zs, символьная группа, 914

А

абсолютные выражения XPath, 82
абсолютный путь доступа, определение, 934
абстрактные типы данных, 894
абстрактные элементы, 894
активные интерфейсы, 36
анонимный тип данных, 882
апострофы, 184
арность, определение, 934
атомарные значения
group-adjacent, атрибут, 267
XPath 2.0, 68
определение, 934
последовательность, 248
сравнение, 109
атрибуты
XPath, 67
<xsl:sort>, 243
выбор, 83
методы вывода, 52
общие, 400
объявление, 224
пустые элементы, 878
элементы, 879

Б

базовые URI-адреса
document(), функция, 287
определение, 934
базовый процессор XSLT 2.0, 131
базовый тип данных
xs:anyType, тип данных, 95
блоки, 915
буквы, символьные группы, 913

В

верхнего уровня, элементы
общие сведения, 55
определение, 944
вещественный тип данных, 93, 582
вложенные элементы, 26
возможности
XPath 2.0, 67
XSLT, 20
время, форматирование, 159, 923
встроенные типы данных XML Schema, 68

выбор
атрибуты, 83
текст из элементов, 83
узлы комментариев, 84
вывод, 141
<xsl:copy> и <xsl:copy-of>, 161
дата и время, 159
несколько файлов, 318
нумерация частей документа, 147
пропуски, 167
текст, 141
числа, 156
выражения
комментарии, 129
сравнение, 107
типы данных, возвращаемые в XPath 1.0, 92
выходные документы, 556

Г

генератор случайных чисел, 369
глобальные параметры, 186
Гринвич, время, 94
группы
XSLT 2.0, 262, 927
типы данных, 883
узлы, 252
группы подстановки, 893

Д

даты, форматирование, 159, 923
двоичные октеты, 94
действительные документы
XML Schema, 30
определение, 935
деление, типы данных, 929
денежные единицы (Sc), символьная группа, 915
дерево результата, определение, 935
диапазонные выражения XPath 2.0, 118
динамические функции, определение, 369
динамический контекст
XPath 2.0, 79
общие сведения, 80
определение, 935
динамический тип, определение, 122

З

закрывающие символы (Pe), символьная группа, 914
замена с использованием рекурсии, 203
знаки препинания, символьные группы, 914
значения
key(), 226
атрибутов, 26
вещественные, 92
группировка, 293
даты, 94
сравнение с использованием предикатов, 218
значения по умолчанию
атрибуты, 71
параметры, 186

И

имена, вызов шаблонов, 181
импортирование
XML Schema, 897
таблицы стилей, 54
инструкции по обработке, 935
интерактивность, SAX, 35
интерфейсы, программные, 32
история XML и XSLT, 19
итераторы, для последовательностей, 115
итерация, 21

К

календари, 925
карты символов, <xsl:character-map>, 429
квантификаторы, 906
Кэй, Майкл, 11
ключевые слова XPath, 584
ключи, 226
<xsl:key>, 55
определение, 935
коды символов, 936
коды форматирования, 921
дата и время, 923
числа, 922
коллекции, контекст XPath, 579
команды запросов, процессор Xalan, 376
комментарии в выражениях XPath, 129
компоненты путей доступа, определение, 936

конечные знаки цитирования (Pf), символьная группа, 914
конечные теги, разбор, 30
конструкторы
XPath 2.0, 119
определение, 943
список, 592
контекстные узлы
XPath, контекст, 578
определение, 937
контекстные функции, список, 592
корневые узлы
ancestor, ось, 88
общие сведения, 70
определение, 574
корневые шаблоны, как метод main, 50
корневые элементы, 133
корректные документы
определение, 937
сравнение с действительными, 30
круговые диаграммы, 370

Л

литеральный элемент результата, определение, 937
логические значения
<xsl:if>, 176
логические операторы XPath, 107
логические функции, список, 591
логический тип данных, 92, 581
локальные имена
определение, 937
при поиске узлов, 71

М

математические операции
без рекурсии, 297
ошибки, 929
математические функции, 369
метасимволы, 910
метод вывода, элемент <xsl:output>, 52, 513
минимальные квантификаторы, 906
многострочный режим, 402
модель данных, 68
модификаторы (Sk), символьная группа, 915
модификаторы представления, 924
Мюнха, метод, 258

Н

набор узлов, 92, 580
наборы атрибутов
<xsl:attribute-set>, 421
определение, 937
начальные знаки цитирования (Pi), символьная группа, 914
неопределенные параметры, в XSLT 1.0, 190
непротяженные знаки (Mn), символьная группа, 914
неразобранные сущности, 938
неразобранный текст, 938
нумерация, 147

О

область видимости
параметры, 186
переменные, 197
элементы в объявлениях пространств имен, 79
обратные ссылки
общие сведения, 909
общие функции, 369
обязательные параметры в XSLT 2.0, 192
объединение документов XML, 279
collection(), 307
doc() и doc-available(), 305
document(), 279
unparsed-text() и unparsed-text-available(), 309
XSLT 2.0, 295
группировка, 293
таблицы отображения, 290
объявление
атрибуты, 224
элементы, 878
объявления пространств имен
контекст XPath, 579
определение, 938
элементы, 79
огораживающие знаки (Me), символьная группа, 914
ограничение, при создании типов данных, 887
одноуровневые узлы, определение, 938
операторы
XPath, 96, 584
XQuery 1.0 and XPath 2.0, 37

приоритет в XPath, 589
операторы множеств XPath 2.0, 124
операторы сравнения
атомарные значения, 109
операторы узлов XPath 2.0, 127
определение, типы данных, 882
оси, 938
пути доступа, 87
список в XPath, 87
откат, 314
относительные выражения XPath, 82
относительные ссылки, 287
ошибки
version, атрибут, 400
<xsl:call-template> с неопределенным параметром, 930
XslCompiledTransform, 338
вызов функций с несколькими узлами, 930
импортированные шаблоны, 407
одноименные переменные, 563
переход, 931
префиксы элементов расширения, 319
при вызове generate-id(), 233

П

параметры, 183
<xsl:param>, 55
XSLT 2.0, 190
определение, 939
определение в шаблонах, 183
передача шаблонам, 184
пассивные интерфейсы, 36
переменные
контекст, 80
общие сведения, 21, 199
определение, 939
ошибки, 563
содержащие узлы, 565
подвыражения, 904
позиция контекста, 939
контекст XPath, 578
полиморфизм, при вызове шаблонов, 181
полные имена, 939
пользовательские схемы упорядочения, 323
порядок документа, 939
последовательности
XPath 2.0, 68, 75
<xsl:perform-sort>, 250, 526
итераторы, 115
определение, 939
передача generate-id(), 232
сравнение, 110
потомки, определение, 939
правила
DTD, 28
документы XML, 24
шаблоны, 52
предикатные выражения
выбор первого узла, 232
определение, 940
предикаты
XPath, 89
<xsl:for-each>, элемент, 248
сравнение значений, 218
предки, определение, 940
представление, отделение от содержимого, 24
предупреждения, 52
преобразование типа, 940
префиксы пространства имен, 940
привязка данных, 36
примитивные типы, 940
приоритет, 53
проверка типов XSLT 2.0, 930
пропуски
<xsl:analyze-string>, 402
<xsl:preserve-space>, 530
в HTML, 142
в деревьях DOM, 35
обработка, 167
определение, 940
элементы, 55
пространства имен
XML Schema, 899
<xsl:function>, 316
общие сведения, 40
определение, 940
по умолчанию, 579
правила, 31
расширения, 315
простые выражения, 903
пустые последовательности, определение, 940
пустые элементы, создание, 878
пути доступа, 78
метасимволы, 85
определение, 941
оси, 87

относительные и абсолютные выражения, 82
предикаты, 89
простые, 81

P

разбор
документы XML, 25
таблицы стилей, 49
узлы, состоящие из одних пропусков, 170
разделители, символьные группы, 914
разделитель строк (Zl), символьная группа, 914
размер контекста
контекст XPath, 579
определение, 79, 941
расширение XSLT, 313
EXSLT, 369
базы данных, 374
несколько выходных файлов, 318
пользовательское упорядочение, 323
пример с фотоальбомом, 378
создание новых функций, 316
расширения, функции, 943
расширенные имена, определение, 941
регулярные выражения, 903
XSLT 2.0 and XPath 2.0, 928
квантификаторы, 906
метасимволы, 910
минимальные квантификаторы, 906
обратные ссылки, 909
общие сведения, 941
подвыражения, 904
простые выражения, 903
режимы обработки, 907
символьные группы, 913
якори, 908
режимы обработки, 907
рекурсивные шаблоны, 285
рекурсия
XSLT 2.0 and XPath 2.0, 927
объединение документов XML, 285
определение, 21
правила шаблонов, 53
применение, 201
процессоры XSLT 2.0, 285
родитель, 941

C

сериализация, 38
сигнатуры функций, 80
символы форматирования (Cf), символьная группа, 915
символьные группы, 913
синтаксис
XPath, 67, 129
XQuery 1.0, 38
xsl:sort, 237
группировка, 241
туннельные параметры, 199
сложение (+), оператор, 584
сложение, оператор XPath, 96
события в SAX, 35
содержимое, отделение от представления, 24
соединители (Pc), символьная группа, 914
сортировка
document(), функция, 288
данные, 237
пользовательские схемы упорядочения, 323
составные типы, 941
сравнение
атомарные значения, 109
выражения, 107
последовательности, 110
текст, 324
ссылки, 212
в неструктурированных документах, 230
типы ID, IDREF и IDREFS, 212
стандарты
XML, 38
XSLT, 36
таблицы стилей, 20
статические методы, 340
статические ошибки, 941
статический контекст
XPath 2.0, 79
общие сведения, 79
определение, 941
статический тип данных, 122
строковые данные, 92, 583
преобразование к логическим значениям, 177
строковые функции, список, 595

схемосовместимые процессоры
XSLT 2.0, 131
схемы упорядочения по умолчанию, URI, 401

Т

таблицы отображения
<xsl:function>, 298
объединение документов XML, 290
таблицы стилей
версии 1.0 и 2.0, 318
импортирование, 54
как документы XML, 20
определение, 942
отладка, 57
пример Hello World, 46
связь с документами XML, 40
стандарты, 20, 38
текст
в узлах CDATA, 67
включение пропусков в вывод, 172
выбор из элементов, 83
построение, 141
сравнение, 324
элементы, 880
текстовые узлы
встроенные правила, 54
общие сведения, 72
определение, 575, 942
текущие узлы, определение, 942
тесты узлов
XPath, 73
определение, 942
типы данных
XML ID, IDREF и IDREFS, 212
XPath, 91, 580
XPath 2.0, 928
XSLT 2.0, 32, 193, 928
определение, 882
параметры, 190
преобразование к логическим значениям, 176
при делении, 929
типы узлов в XPath, 69
тождественное преобразование, 162
точки связывания, создание, 222
туннельные параметры, 194, 942

У

узлы
<xsl:apply-templates>, 410
вызов generate-id(), 232
контекст, 49
обработка абсолютных выражений, 82
последовательность, 248
сложение, 98
сравнение, 124
тесты, 576
типы в XPath, 69, 574
узлы атрибутов
встроенные правила, 54
общие сведения, 71
определение, 575, 942
узлы документов
встроенные правила, 53
определение, 943
узлы инструкций по обработке
встроенные правила, 54
выбор, 84
общие сведения, 73
определение, 575, 943
узлы комментариев, 72
встроенные правила, 54
выбор, 84
определение, 575, 943
узлы пространств имен
встроенные правила, 54
выбор, 84
общие сведения, 73
определение, 943
узлы элементов
встроенные правила, 53
общие сведения, 70
определение, 574, 943
умножение, оператор, 584
упорядочение по кодам символов, 943
управляющие символы (Cc), символьная группа, 915
условные выражения XPath 2.0, 113
установка процессоров XSLT, 41

Ф

фигурные скобки ({}), в регулярных выражениях, 419
формальная семантика XQuery 1.0 XPath 2.0, 37

форматирование
дата и время, 159
числа, 156
фотоальбом, 378
функции
QName, список, 594
XQuery 1.0 and XPath 2.0, 38
в предикатах, 89
временных промежутков, список, 592
вызов с несколькими узлами, 930
группировки, список, 592
даты и времени, список, 592
доступа, список, 591
контекст, 80
контекст XPath, 579
множеств, 369
наборов узлов, список, 594
общие сведения, 22
по категориям, 591
последовательностей, список, 594
регулярных выражений, список, 594
создание, 316
узлов, список, 593
упорядочения, список, 596
функциональные языки, связь с XSLT, 21

Ц

циклы
имитация, 206
использование, 21
цифры (Nd), символьная группа, 914

Ч

частичная проверка по схеме, 30
числа
\d, 913
в предикатах, 89
коды форматирования, 922
ошибки в операциях, 929
форматирование, 156
числовые типы
<number>, 123
определение, 92
ошибки в операциях, 929
преобразование к логическим значениям, 176
числовые функции, список, 593

Ш

шаблоны
вызов по имени, 181
выражения XPath, 82
имитация циклов, 206
определение, 944
определение параметров, 183
передача параметров, 184
правила, 53
шаблоны значений атрибутов
определение, 944
упорядочение, 386

Э

экранирование, определение, 944
элементы
верхнего уровня, 55
видимость объявлений пространств имен, 79
выбор текста, 83
дочерние, 71
и теги, 30
объявление, 878
правила документов XML, 25
пропуски, 55
пустые, 878
с текстом, 880
со смешанным содержимым, 881
элементы документов
определение, 26, 944
элементы программной логики, 176
<xsl:choose>, 178
<xsl:for-each>, 180
<xsl:if>, 176

Ю

Юникод, 430

Я

якори, 908

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 978-5-93286-150-9, название «XSLT, 2-е издание» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.