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.

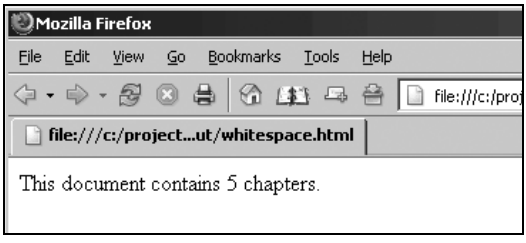

***Рис. 4.1.** HTML нормализует пропуски перед отображением документа*

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.

`Y0001`

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.