

**Московский государственный технический университет  
имени Н.Э. Баумана**

**Кафедра «Системы обработки информации и управления»**

к.т.н. профессор Э.Н. Самохвалов

к.т.н. доцент Г.И. Ревунков

к.т.н. доцент Ю.Е. Гапанюк

**Методические указания  
к лабораторным работам по курсу  
XML – технологии  
Часть 1  
(4 семестр)**

**Москва**

**2012**

# СОДЕРЖАНИЕ

<b>1</b>	<b>ТЕОРЕТИЧЕСКАЯ ЧАСТЬ.....</b>	<b>3</b>
1.1	ВВЕДЕНИЕ В ЯЗЫК XML.....	3
1.2	ОПИСАНИЕ СТРУКТУР ДАННЫХ С ПОМОЩЬЮ XML.....	4
1.3	ОСНОВЫ ЯЗЫКА XPATH.....	7
1.4	ВВЕДЕНИЕ В XSLT .....	17
1.4.1	<i>XSLT-преобразования с фиксированной структурой.....</i>	<i>17</i>
1.4.2	<i>XSLT-преобразования с адаптируемой структурой.....</i>	<i>24</i>
<b>2</b>	<b>УСЛОВИЯ ЛАБОРАТОРНЫХ РАБОТ.....</b>	<b>31</b>
2.1	ОПИСАНИЕ СТРУКТУР ДАННЫХ С ИСПОЛЬЗОВАНИЕМ XML .....	31
2.2	РАЗРАБОТКА XPATH-ЗАПРОСОВ .....	31
2.3	РАЗРАБОТКА XSLT-ПРЕОБРАЗОВАНИЯ С ФИКСИРОВАННОЙ СТРУКТУРОЙ .....	31
2.4	РАЗРАБОТКА XSLT-ПРЕОБРАЗОВАНИЯ С АДАПТИРУЕМОЙ СТРУКТУРОЙ.....	31
<b>3</b>	<b>ТРЕБОВАНИЯ К ОТЧЕТАМ .....</b>	<b>31</b>
<b>4</b>	<b>КОНТРОЛЬНЫЕ ВОПРОСЫ .....</b>	<b>32</b>
<b>5</b>	<b>ЛИТЕРАТУРА .....</b>	<b>32</b>

# 1 Теоретическая часть

## 1.1 Введение в язык XML

XML является упрощенной версией языка SGML (Standard Generalized Markup Language, стандартный обобщенный язык разметки). SGML был утвержден ISO в качестве стандарта в 1986 году. SGML предназначен для создания других языков разметки, он определяет допустимый набор тэгов, их атрибуты и внутреннюю структуру документа. Контроль над правильностью использования тэгов осуществляется при помощи специального набора правил, называемых DTD- описаниями, которые используются при разборе документа.

Из-за своей сложности SGML использовался, в основном, для описания синтаксиса других языков разметки (наиболее известным из которых является HTML), и немногие приложения работали с SGML- документами напрямую.

XML является подмножеством SGML. То есть XML не содержит фиксированного набора тэгов и предназначен для создания языков разметки, подобных HTML. В XML используются только те возможности SGML, которые реально необходимы в Web.

XML обеспечивает ряд функциональных возможностей, которые отсутствуют в HTML:

- Позволяет разработчикам определять собственные тэги и атрибуты так, как это позволяет делать SGML.
- Предоставляет возможность проверки действительности структуры документов во время их обработки с помощью DTD или схем данных.

Существует два основных варианта использования XML:

1. Моделирование предметных областей и создание языков разметки на основе XML.

На основе XML создаются такие новые языки разметки, как:

- XHTML – расширяемый вариант HTML

- MathML (Mathematical Markup Language) - Формат описания математических формул.
- CML (Chemical Markup Language) - Формат описания химических формул.
- WML (Wireless Markup Language) - Вариант HTML для сотовых телефонов, используемый в WAP-технологии.
- SVG (Scalable Vector Graphics) - язык описания двухмерных векторных изображений.

2. Использование XML в качестве обменного формата в гетерогенных компьютерных системах (технология веб-сервисов).

## **1.2 Описание структур данных с помощью XML**

### **Множество:**

```
<?xml version="1.0" encoding="Windows-1251"?>
<!-- Множество -->
<Множество>
    <Элемент_множества id="1">
        <Параметры/>
    </Элемент_множества>
    <Элемент_множества id="2">
        <Параметры/>
    </Элемент_множества>
    <!-- . . . -->
    <Элемент_множества id="N">
        <Параметры/>
    </Элемент_множества>
</Множество>
```

### **Массив:**

```
<?xml version="1.0" encoding="Windows-1251"?>
<!-- Массив -->
<Массив>
    <Элемент_массива номер="1">
```

```

        <Значение/>
    </Элемент_массива>
    <Элемент_массива номер="2">
        <Значение/>
    </Элемент_массива>
    <!-- . . . -->
    <Элемент_массива номер="n">
        <Значение/>
    </Элемент_массива>
</Массив>

```

Описание массива и множества не отличаются друг от друга. Элементы массива могут быть упорядочены по следованию друг за другом или по значению атрибута «номер». В множестве упорядоченность не учитывается.

### **Дерево:**

```

<?xml version="1.0" encoding="Windows-1251"?>
<!-- Дерево -->
<Дерево>
    <Ветвь_11>
        <Лист/>
        <Ветвь_21>
            <Лист/>
            <Лист/>
            <Лист/>
        </Ветвь_21>
        <Ветвь_22>
            <Лист/>
            <Лист/>
        </Ветвь_22>
    </Ветвь_11>
    <Ветвь_12>
        <Лист/>
    </Ветвь_12>
</Дерево>

```

Древовидные структуры являются «естественными» для модели данных XML.

### **Граф (связи задаются в вершинах):**

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

```
<Граф>
```

```
  <Вершина id="1">
```

```
    <Данные_о_вершине/>
```

```
    <Выходы>
```

```
      <Выход вершина="2"/>
```

```
      <Выход вершина="3"/>
```

```
    </Выходы>
```

```
  </Вершина>
```

```
  <Вершина id="2">
```

```
    <Данные_о_вершине/>
```

```
    <Выходы>
```

```
      <Выход вершина="1">
```

```
        <Данные_о_связи/>
```

```
      </Выход>
```

```
    </Выходы>
```

```
  </Вершина>
```

```
  <Вершина id="3">
```

```
    <Данные_о_вершине/>
```

```
  </Вершина>
```

```
</Граф>
```

### **Граф (вершины и связи задаются в отдельных секциях документа):**

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

```
<Граф>
```

```
  <Вершины>
```

```

    <Вершина id="1">
        <Данные_о_вершине/>
    </Вершина>
    <Вершина id="2"/>
    <Вершина id="3"/>
</Вершины>
<Связи>
    <Связь вершина_1="1" вершина_2="2">
        <Данные_о_связи/>
    </Связь>
    <Связь вершина_1="1" вершина_2="3"/>
    <Связь вершина_1="2" вершина_2="1"/>
</Связи>
</Граф>

```

Такое описание удобно использовать как для ориентированного, так и для неориентированного графа.

Таким образом, с помощью модели данных XML достаточно просто описывать различные структуры данных.

### **1.3 Основы языка XPath**

XPath – это набор синтаксических правил для адресации частей XML-документа. Язык XPath используется в технологии XSLT и в некоторых XML-ориентированных базах данных.

Главная задача XPath-запроса (XPath-выражения) – найти нужный фрагмент (элемент, атрибут) документа XML.

XPath-выражение представляет собой строку, в которой адресуется фрагмент XML-документа. XPath-выражение похоже на запись пути в файловой системе, только вместо названий каталогов и файлов используются названия элементов и атрибутов в документе.

#### **Документ XML для изучения XPath-выражений:**

```

<?xml version="1.0"?>
<A>
    <B b="b1">B1</B>

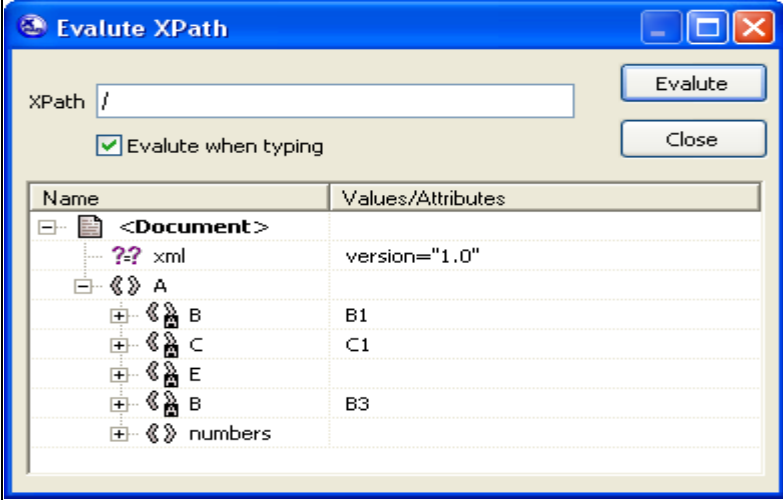
```

```

<C a="a1">C1</C>
<E e="e1">
  <B a="a2">B2</B>
  <G>G</G>
</E>
<B b="b3">B3</B>
<numbers>
  <number id="1">2</number>
  <number id="2">4</number>
  <number id="3">8</number>
</numbers>
</A>

```

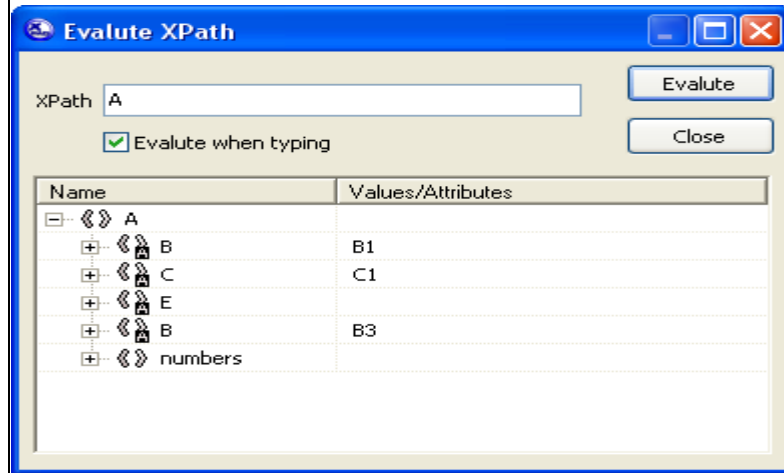
Примеры основных выражений XPath и результаты их выполнения приведены в следующей таблице.

XPath-выражение	Результат выполнения																		
/	<p>Корневой элемент документа.</p>  <p>The screenshot shows a dialog box titled "Evaluate XPath". The XPath input field contains the expression "/". Below the input field, there is a checked checkbox labeled "Evaluate when typing". To the right of the input field are "Evaluate" and "Close" buttons. The main area of the dialog is a table with two columns: "Name" and "Values/Attributes". The table content is as follows:</p> <table border="1" data-bbox="536 1240 1275 1532"> <thead> <tr> <th>Name</th> <th>Values/Attributes</th> </tr> </thead> <tbody> <tr> <td>&lt;Document&gt;</td> <td></td> </tr> <tr> <td>xml</td> <td>version="1.0"</td> </tr> <tr> <td>A</td> <td></td> </tr> <tr> <td>B</td> <td>B1</td> </tr> <tr> <td>C</td> <td>C1</td> </tr> <tr> <td>E</td> <td></td> </tr> <tr> <td>B</td> <td>B3</td> </tr> <tr> <td>numbers</td> <td></td> </tr> </tbody> </table>	Name	Values/Attributes	<Document>		xml	version="1.0"	A		B	B1	C	C1	E		B	B3	numbers	
Name	Values/Attributes																		
<Document>																			
xml	version="1.0"																		
A																			
B	B1																		
C	C1																		
E																			
B	B3																		
numbers																			



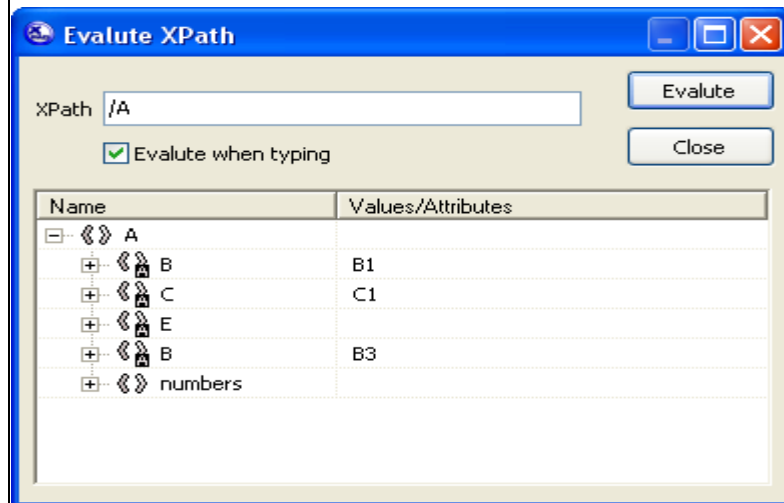
A

Элемент А.



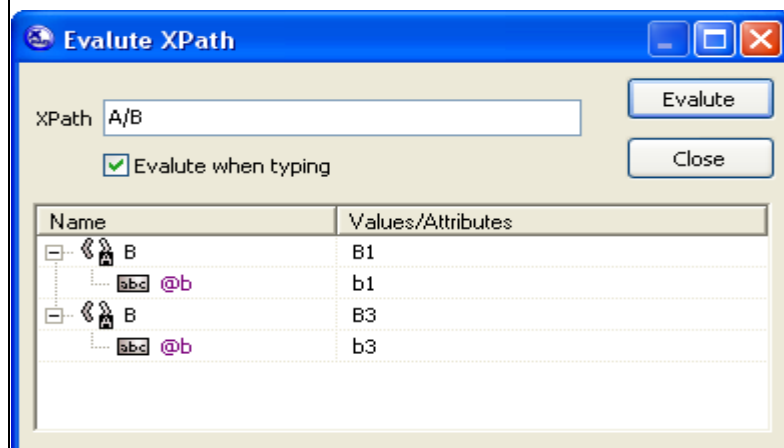
/A

Элемент А, вложенный в корневой элемент. Результат выполнения такой же, как в предыдущем запросе.



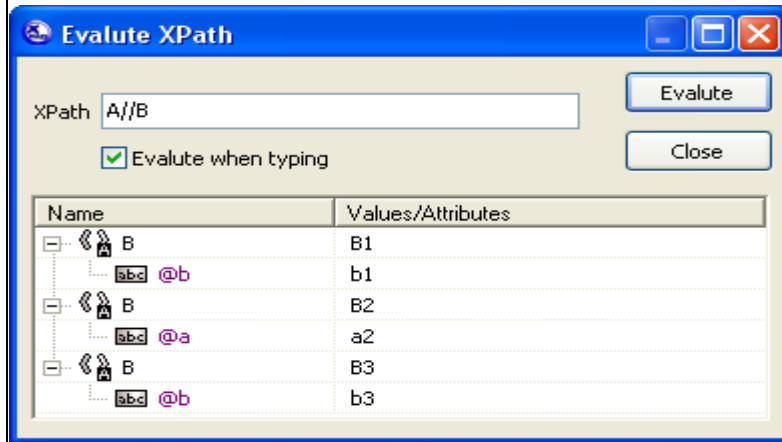
A/B

Элемент В, вложенный в элемент А.



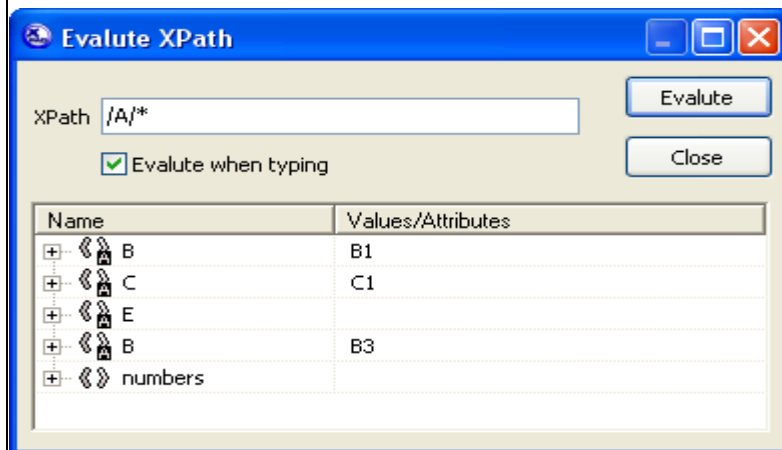
A//B

Элемент B, вложенный на любой глубине в элемент A. Обратите внимание, что элемент B со значением B2 не вложен непосредственно в элемент A.



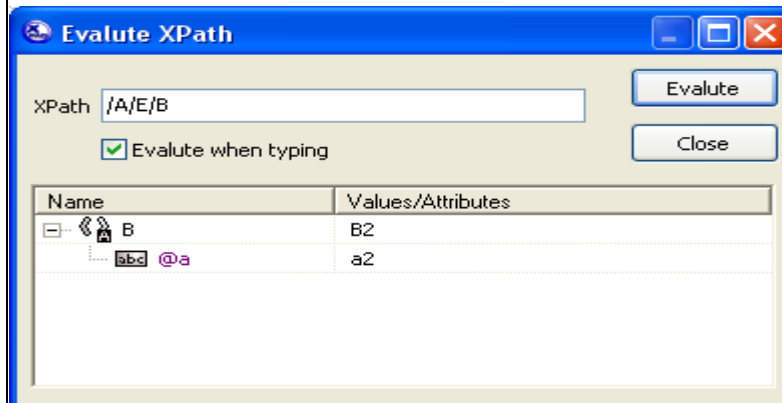
/A/\*

Любой элемент, вложенный в A, вложенный в корневой элемент.



/A/E/B

Элемент B, вложенный в E, вложенный в A, вложенный в корневой элемент.



A/B | A/C

Все элементы В или С, вложенные в А, символ « | » – оператор объединения множеств.

The screenshot shows the 'Evaluate XPath' dialog box with the XPath expression 'A/B | A/C' entered. The 'Evaluate when typing' checkbox is checked. The results table is as follows:

Name	Values/Attributes
B	B1
abc @b	b1
B	B3
abc @b	b3
C	C1
abc @a	a1

A/\*\*

Все элементы, вложенные в А на любой глубине.

The screenshot shows the 'Evaluate XPath' dialog box with the XPath expression 'A/\*\*' entered. The 'Evaluate when typing' checkbox is checked. The results table is as follows:

Name	Values/Attributes
B	B1
C	C1
E	
B	B2
G	G
B	B3

//B/@b

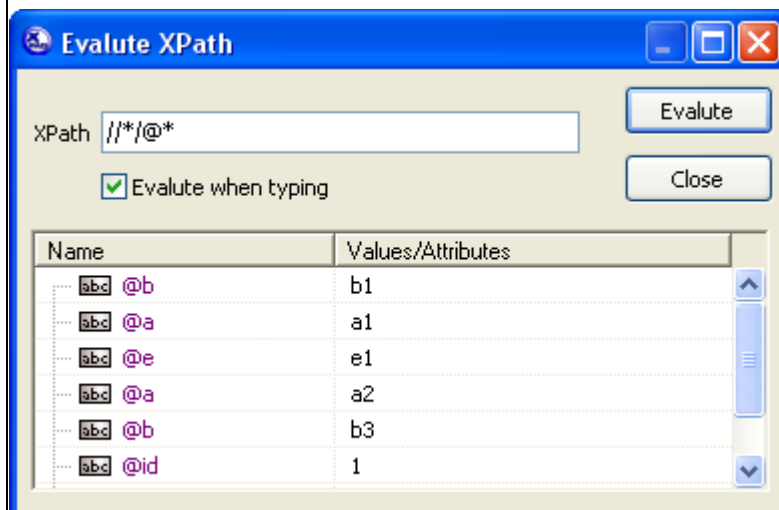
Атрибуты b, вложенные в элементы В. Символ @ означает, что за ним следует название атрибута, а не элемента.

The screenshot shows the 'Evaluate XPath' dialog box with the XPath expression '//B/@b' entered. The 'Evaluate when typing' checkbox is checked. The results table is as follows:

Name	Values/Attributes
abc @b	b1
abc @b	b3

`//*[@*]`

Все атрибуты всех элементов документа.



Обратите внимание, что если в запросе используется `//`, то такой запрос может «подтягивать» элементы на более высокий уровень иерархии. Например, в запросе `A//*`, элементы E и G возвращаются на одном уровне иерархии, хотя в соответствии с примером элемент G вложен в E.

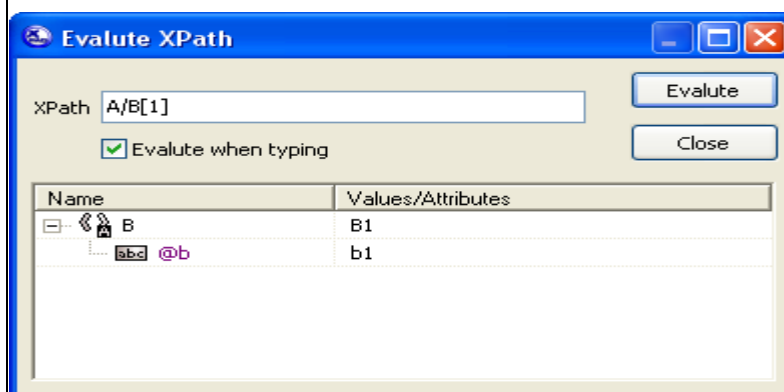
Если после названия элемента стоит выражение в квадратных скобках, то это означает, что к элементу применяется фильтр. Выбираются не все элементы, а только те, которые удовлетворяют фильтру.

В качестве фильтра может быть задано число, которое определяет номер элемента. Это похоже на использование индексов в массиве. Или в качестве фильтра может быть задано условие поиска.

`A/B[1]`

Первый элемент B, вложенный в A.

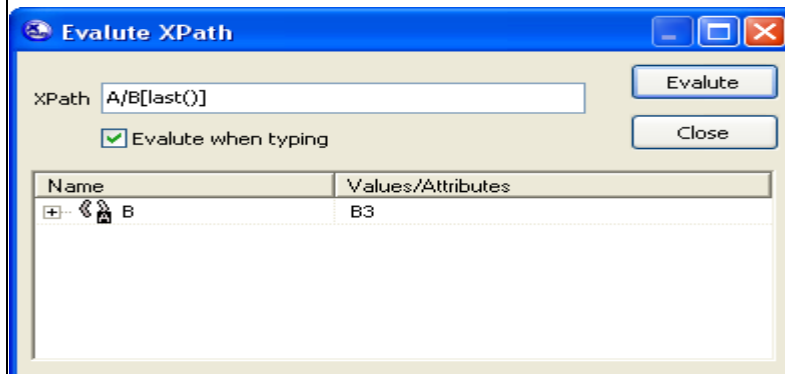
(В некоторых реализациях XPath нумерация начинается с 0, в некоторых с 1).



A/B[last()]

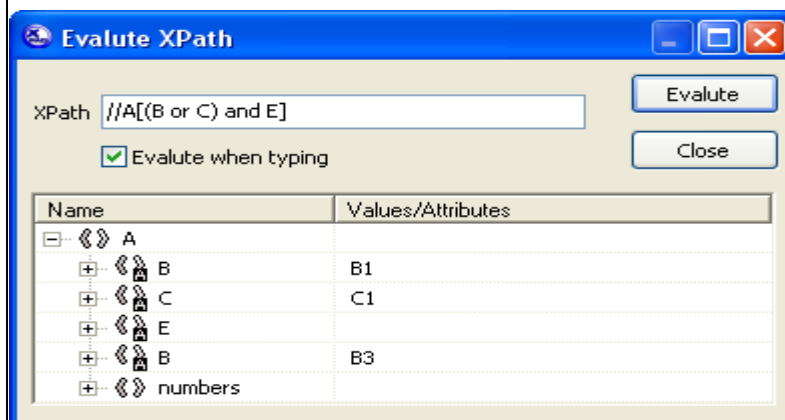
Последний элемент В, вложенный в А.

Функция last() возвращает индекс последнего элемента в последовательности.



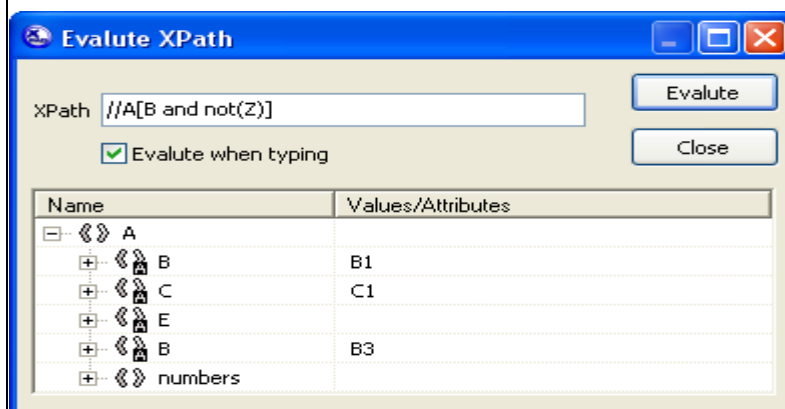
//A[(B or C) and E]

Поиск такого элемента А, в который вложены элементы В или С и вложен элемент Е. Значения элементов В, С, Е в данном запросе не важны. Здесь проверяется только наличие вложенных элементов.



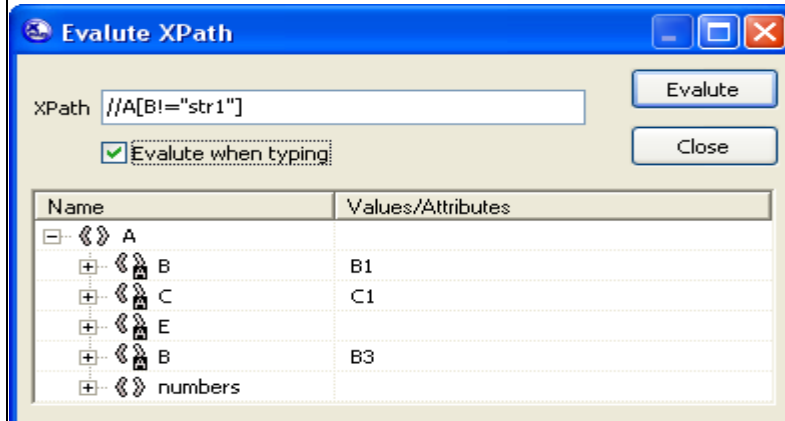
//A[B and not(Z)]

Поиск такого элемента А, в который вложен В и не вложен Z.



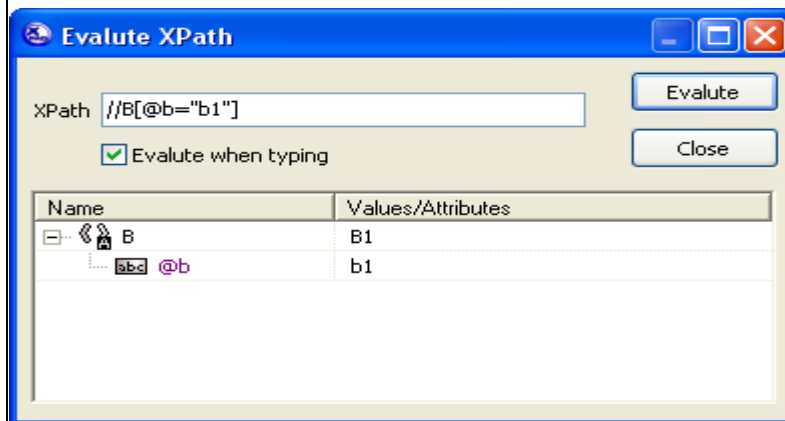
//A[B!="str1"]

Поиск такого элемента А, у которого есть вложенный элемент В не равный «str1».



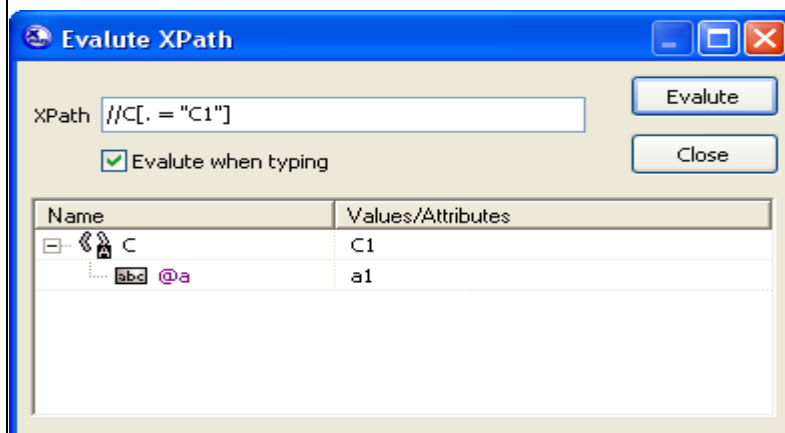
//B[@b="b1"]

Поиск элемента В, у которого есть атрибут b=b1. Символ @ означает, что за ним следует название атрибута, а не элемента.



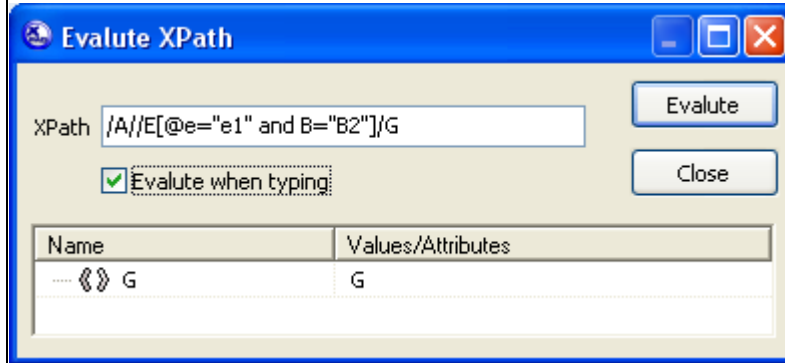
//C[.="C1"]

Поиск элемента С = C1. Точка означает, что проверяется значение текущего элемента.



`/A//E[@e="e1" and B="B2"]/G`

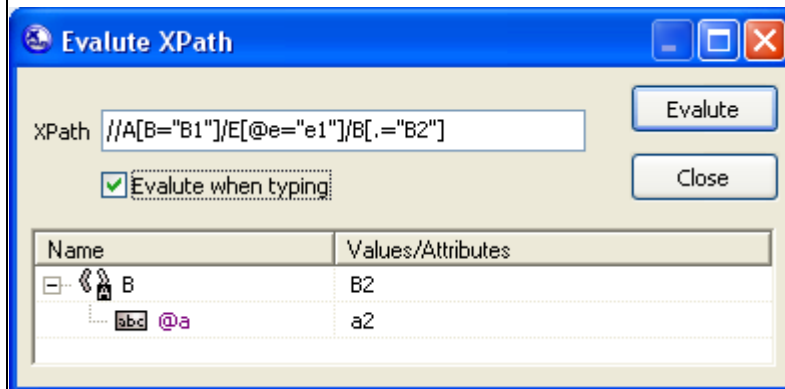
Все элементы G, непосредственно вложенные в E. Элемент E вложен в A на любой глубине. Элемент A непосредственно вложен в корневой элемент. Элемент E должен содержать атрибут e=e1 и вложенный элемент B=B2.

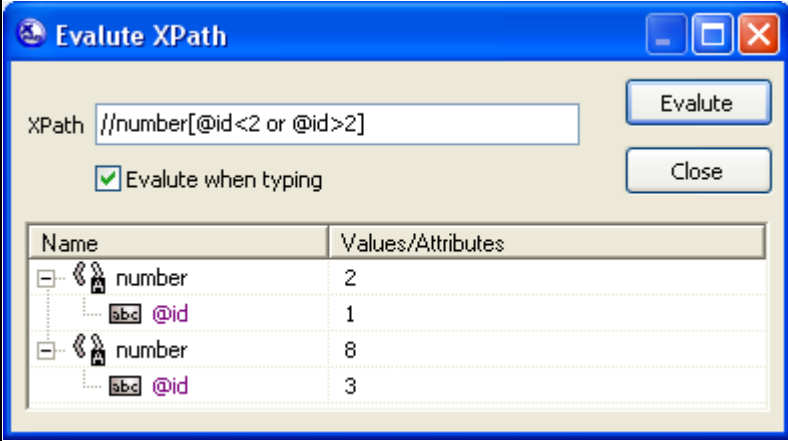
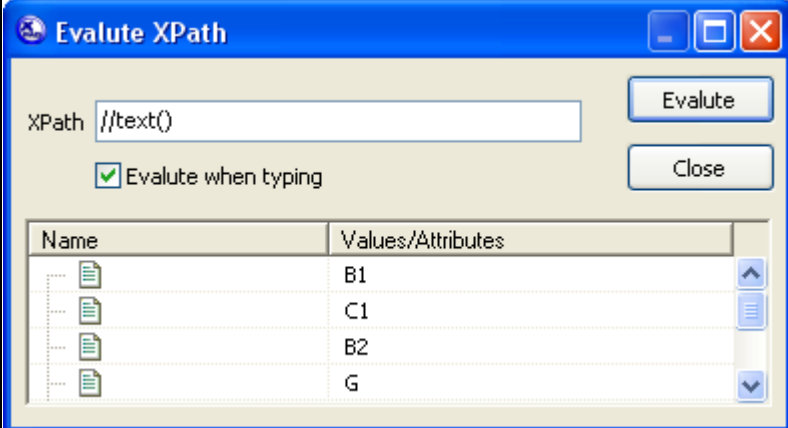


`//A[B="B1"]/E[@e="e1"]/B[.="B2"]`

Элемент B=B2 должен быть вложен в E. Элемент E должен содержать атрибут e=e1. Элемент E должен быть вложен в элемент A. Элемент A должен содержать вложенный элемент B=B1.

Этот пример показывает, что фильтры могут применяться к элементам на любой глубине вложенности и фильтрами могут быть снабжены несколько элементов в запросе.



<p>//number[@id&lt;2 or @id&gt;2]</p>	<p>Поиск элементов number, у которых атрибут id&lt;2 или атрибут id&gt;2.</p> 
<p>//text()</p>	<p>Все текстовые узлы документа</p> 

Для сравнения могут быть использованы следующие операторы: = (равенство), != (неравенство), <, <=, >, >=.

В некоторых реализациях XPath знаки <, >, заменяются на соответствующие ссылки на символы &lt; и &gt;. Тогда операторы сравнения <=, >= будут записаны как &lt;=, &gt;=.

Для выборки конструкций определенного типа (иногда их называют узлами документа) можно использовать следующие функции:

- text() – любой текстовый узел.
- node() – любой узел, который не является атрибутом и корневым элементом.
- comment() – комментарий.
- processing-instruction() – инструкция обработки.



## **1.4 Введение в XSLT**

Для преобразования XML–документов в другие форматы была разработана технология XSLT.

XSLT расшифровывается как Extensible Stylesheet Language Transformations, расширяемый язык стилевых преобразований.

Как правило, с помощью XSLT выполняют три варианта преобразований:

1. Из XML в HTML. Этот вариант часто применяется в Web-приложениях.
2. Из XML в другой словарь XML (в другой набор тэгов). Этот вариант называют преобразованием словарей XML.
3. Из XML в текстовый формат, например в CSV (comma separated values – формат, в котором разделителями являются запяты).

Единственная задача, которую нельзя решить напрямую с помощью технологии XSLT – это преобразование из XML в двоичный формат. Для решения этой задачи можно использовать технологию расширений XSLT, когда из XSLT-преобразования вызывается исполняемый код.

XSLT-преобразование осуществляется специализированной программой или библиотекой, которая называется XSLT-процессор.

Принцип обработки XML-документов заключается в следующем: при разборе XSLT-документа XSLT-процессор обрабатывает инструкции этого языка и каждому элементу, найденному в XML-дереве, ставит в соответствие набор тэгов HTML, XML или текст, которые определяют форматирование этого элемента.

Инструкции XSLT определяют точное положение элемента XML в документе с помощью XPath-выражений, поэтому существует возможность применять различные стили оформления к элементам с одинаковыми названиями, в зависимости от их положения в документе.

### **1.4.1 XSLT-преобразования с фиксированной структурой**

Рассмотрим простой пример преобразования XSLT, который преобразует XML-документ, посвященный языкам разметки, в HTML-документ.

## Файл XML:

```
<?xml version="1.0" encoding="Windows-1251"?>
<?xml-stylesheet type="text/xsl" href="Example_1.xsl"?>
<!-- Языки разметки -->
<languages>
  <language id="1">
    <name>HTML</name>
    <year>01.01.1990</year>
    <howold>19</howold>
  </language>
  <language id="2">
    <name>XML</name>
    <year>01.01.1998</year>
    <howold>11</howold>
  </language>
  <language id="3">
    <name>SGML</name>
    <year>01.01.1986</year>
    <howold>23</howold>
  </language>
</languages>
```

## Файл XSLT:

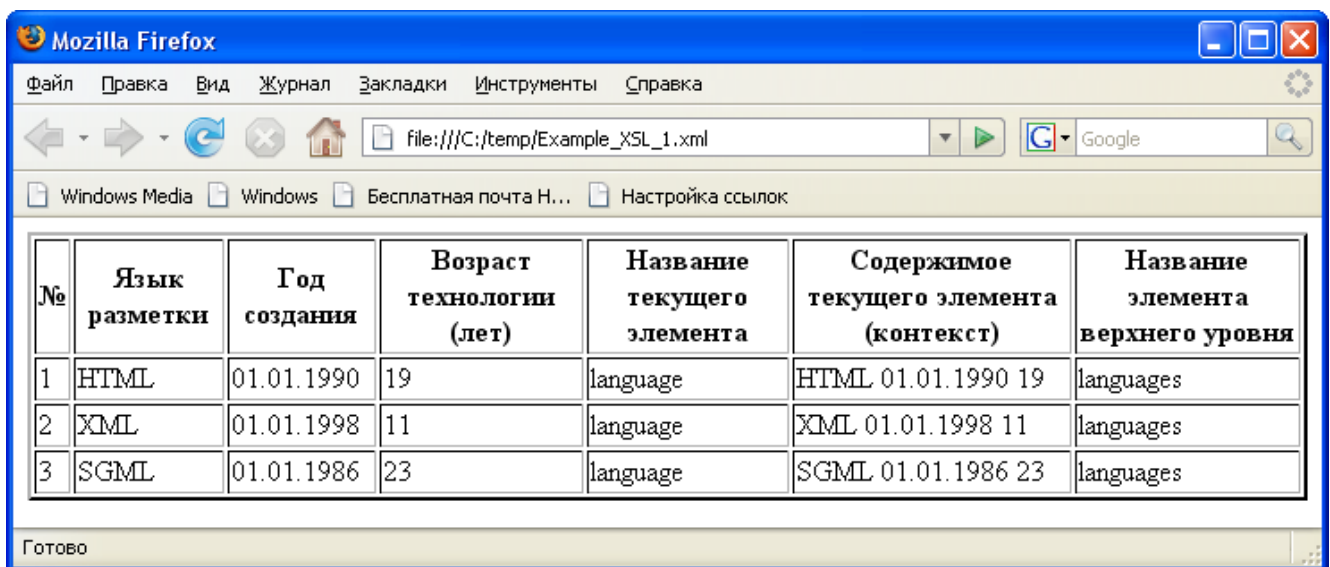
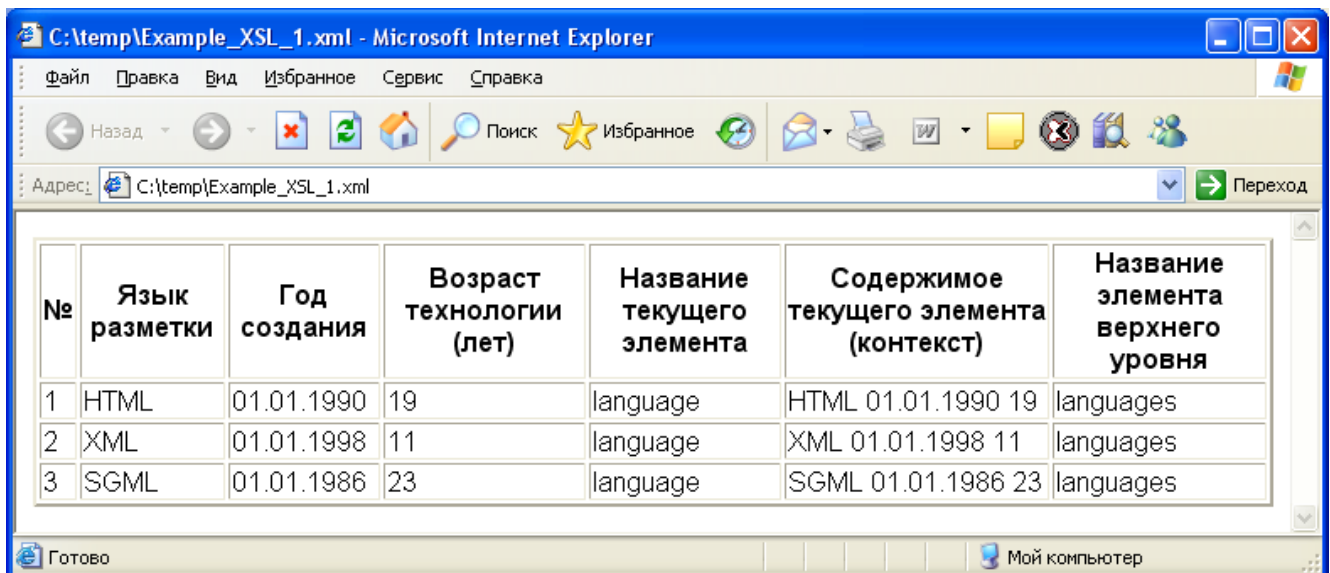
```
<?xml version="1.0" encoding="Windows-1251"?>
<!--XSLT - документ является XML - документом. -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<!-- Описание XSLT - документа -->
<xsl:template match="/">
<!-- Правило обработки корневого элемента XML - документа -->
  <HTML>
  <BODY>
  <TABLE BORDER="2">
  <TR>
    <TH>№</TH>
    <TH>Язык разметки</TH>
```

```

        <TH>Год создания</TH>
        <TH>Возраст технологии (лет)</TH>
        <TH>Название текущего элемента</TH>
        <TH>Содержимое текущего элемента (контекст)</TH>
        <TH>Название элемента верхнего уровня</TH>
    </TR>
    <xsl:for-each select="languages/language">
<!-- Перебор в цикле всех элементов language, вложенных в элемент
languages. -->
        <TR>
            <TD><xsl:value-of select="@id"/></TD>
<!-- Выбор значения атрибута id элемента language -->
            <TD><xsl:value-of select="name"/></TD>
<!-- Выбор значения элемента name, вложенного в элемент language -->
            <TD><xsl:value-of select="year"/></TD>
<!-- Выбор значения элемента year, вложенного в элемент language -->
            <TD><xsl:value-of select="howold"/></TD>
<!-- Выбор значения элемента howold, вложенного в элемент Language
-->
            <TD><xsl:value-of select="name(.)"/></TD>
<!-- Название текущего элемента -->
            <TD><xsl:value-of select="."/></TD>
<!-- Содержимое текущего элемента (контекст) -->
            <TD><xsl:value-of select="name(parent::*)"></TD>
<!-- Название элемента верхнего уровня (также можно использовать
select="name(..)") -->
        </TR>
    </xsl:for-each>
</TABLE>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>

```

**Просмотр результата в браузере:**



В файле XML наиболее интересна вторая строка (инструкция обработки `xml-stylesheet`):

```
<?xml-stylesheet type="text/xsl" href="Example_1.xsl"?>
```

Значение атрибута `type="text/xsl"` указывает на то, что для отображения документа должно быть использовано XSLT-преобразование. В атрибуте `href` задается URI для доступа к файлу XSLT. Расширение файла XSLT обычно «`xsl`».

Если открыть документ с помощью анализатора XML, который поддерживает инструкцию `xml-stylesheet` и может вызывать XSLT-процессор, то XSLT-преобразование будет применено к документу. Такие анализаторы, например, встроены в браузеры Mozilla и Internet Explorer.

Обратим внимание, что преобразование XSLT является документом XML. Этот документ содержит программу на языке XSLT. Команды XSLT задаются в виде XML-элементов. Выделить эти команды просто, так как перед ними стоит префикс « xsl: » (как выяснится позже, замечание по поводу префикса потребует уточнения).

Рассмотрим более подробно текст преобразования XSLT.

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

Преобразование XSLT является XML-документом, поэтому документ начинается с инструкции обработки `<?xml ... ?>`

```
<!--XSLT - документ является XML - документом. -->
```

В преобразовании XSLT используются XML-комментарии `<!-- ... -->`

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

Элемент `stylesheet` является корневым элементом документа и соответствует полной «программе» на языке XSLT. Атрибут `version` с указанием версии обязателен. Атрибут `xmlns:xsl` задает префикс пространства имен.

Вместо элемента `xsl:stylesheet` может быть использован элемент `xsl:transform`.

Эти элементы являются синонимами.

```
<!-- Описание XSLT - документа -->
```

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

Элемент `template` (шаблон) соответствует «процедуре». Элементов `template` в преобразовании может быть несколько, в этом примере он один. Атрибут `match` определяет, какому элементу преобразуемого документа соответствует шаблон. Значение `match="/"` показывает, что шаблон соответствует корневому элементу преобразуемого документа. Шаблон со значением `match="/"` вызывается первым. В общем случае значение атрибута `match` – это XPath-выражение.

```
<!-- Правило обработки корневого элемента XML - документа -->
```

```
<HTML>
```

```
<BODY>
```

```
<TABLE BORDER="2">
```

```
<TR>
```

```
<TH>№</TH>
```

```

<TH>Язык разметки</TH>
<TH>Год создания</TH>
<TH>Возраст технологии (лет)</TH>
<TH>Название текущего элемента</TH>
<TH>Содержимое текущего элемента (контекст)</TH>
<TH>Название элемента верхнего уровня</TH>
</TR>

```

Тэги языка HTML (без префикса «xsl:») просто выводятся в результирующий документ. Это можно рассматривать как оператор вывода. Тэг TR формирует строку таблицы в HTML. Элементы TH формируют заголовки в первой строке таблицы.

```

<xsl:for-each select="languages/language">
<!-- Перебор в цикле всех элементов language, вложенных в элемент
languages. -->

```

Элемент for-each – единственная разновидность циклов в XSLT. Никаких других видов циклов в XSLT нет. В атрибуте select указывается XPath выражение, которое возвращает перебираемые в цикле элементы.

Содержимое элемента for-each (тело цикла) выполняется для каждого узла, который возвращается в атрибуте select.

Поэтому тело цикла будет выполняться три раза, для каждого элемента language.

```

<TR>
  <TD><xsl:value-of select="@id"/></TD>
<!-- Выбор значения атрибута id элемента language -->

```

Обратите внимание, что, попадая внутрь тела цикла, мы «проваливаемся» в контекст, который задается текущим значением атрибута select элемента for-each. Все XPath-выражения внутри тела цикла должны быть контекстными.

Здесь мы формируем строку таблицы с помощью элемента TR. Элемент TD задает ячейку таблицы.

Элемент value-of возвращает значение, задаваемое в виде XPath-выражения в атрибуте select. Обратите внимание, что это XPath-выражение контекстное, оно

возвращает атрибут `id` для текущего элемента `language`, который соответствует текущей итерации цикла.

Атрибут `select` используется и в элементе `for-each`, и в элементе `value-of`. И в том, и в другом случае это XPath-выражение. Однако смысл у них различный.

В элементе `for-each` предполагается, что это выражение возвращает несколько значений. Для каждого значения выполняется итерация цикла.

В элементе `value-of` предполагается, что выражение возвращает единственное значение. Если по ошибке написать неоднозначное выражение, то, как правило, XSLT-процессор возвращает первое значение.

Далее с помощью элемента `value-of` формируются другие ячейки таблицы.

Значение `select="."` возвращает XML-значение для элемента `language`. Но браузер не отображает XML-тэги, а отображает только текстовое содержимое вложенных элементов.

Все XPath-выражения являются контекстными. Выражение `<xsl:value-of select="name(.)"/>` возвращает название текущего элемента (`language`), а выражение `<xsl:value-of select="name(parent::*)/>` возвращает название элемента верхнего уровня (`languages`). Вместо `select="name(parent::*)"` также можно использовать `select="name(..)"`.

```
<TD><xsl:value-of select="name"/></TD>
<!-- Выбор значения элемента name, вложенного в элемент language -->
<TD><xsl:value-of select="year"/></TD>
<!-- Выбор значения элемента year, вложенного в элемент language -->
<TD><xsl:value-of select="howold"/></TD>
<!-- Выбор значения элемента howold, вложенного в элемент language -
-->
<TD><xsl:value-of select="name(.)"/></TD>
<!-- Название текущего элемента -->
<TD><xsl:value-of select="."/></TD>
<!-- Содержимое текущего элемента (контекст) -->
<TD><xsl:value-of select="name(parent::*)" /></TD>
<!-- Название элемента верхнего уровня (также можно использовать
select="name(..)") -->
```

```
</TR>
```

Далее следуют необходимые закрывающие тэги.

```
</xsl:for-each>
```

```
</TABLE>
```

```
</BODY>
```

```
</HTML>
```

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

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

## 1.4.2 XSLT-преобразования с адаптируемой структурой

Рассмотренный шаблон XSLT является шаблоном с фиксированной структурой выходного HTML-документа. Если во входном XML-документе поменять местами элементы `<name>` и `<year>`, то это не приведет к перестановке колонок в таблице.

Рассмотрим XSLT-преобразование, которое не содержит жестко заданную структуру выходного документа, а адаптируется к структуре входного документа.

Адаптивность XSLT-преобразования достигается за счет того, что используются несколько элементов `template`, каждый из которых соответствует элементу входного документа.

Порядок вызова элементов `template` не задается. Они вызываются в такой последовательности, в которой встречаются соответствующие элементы во входном документе.

### Файл XSLT:

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

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

```
<!-- ++++++ -->
```

```
<!-- Шаблон обработки корневого элемента XML - документа -->
```

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

```
  <HTML>
```

```
  <HEAD>
```

```
    <LINK href="met.css" rel="stylesheet" type="text/css"/>
```

```
  </HEAD>
```



```

        <BODY>
        <xsl:apply-templates/>
    </BODY>
</HTML>
</xsl:template>
<!-- ++++++ -->

<!-- ++++++ -->
<!-- Шаблон обработки элемента languages -->
<xsl:template match="languages">
    <!-- Вызов шаблона для элемента language (шаблон вызывается
    необходимое количество раз) -->
        <xsl:apply-templates/>
</xsl:template>
<!-- ++++++ -->

<!-- ++++++ -->
<!-- Шаблон обработки элемента language -->
<xsl:template match="language">

    <!-- Получение значения атрибута id (префикс @ означает атрибут) -->
        <B>Номер: <xsl:value-of select="@id"/></B><BR/>

    <!-- Вызов шаблонов для элементов name, year и howold -->
        <xsl:apply-templates/>
        <HR/>
</xsl:template>
<!-- ++++++ -->

<!-- ++++++ -->
<!-- Шаблон обработки элемента name -->
<xsl:template match="name">
    Наименование языка: <B><xsl:value-of select="."/></B><BR/>
    <!-- select="." - получение значения текущего элемента -->
</xsl:template>

```

```

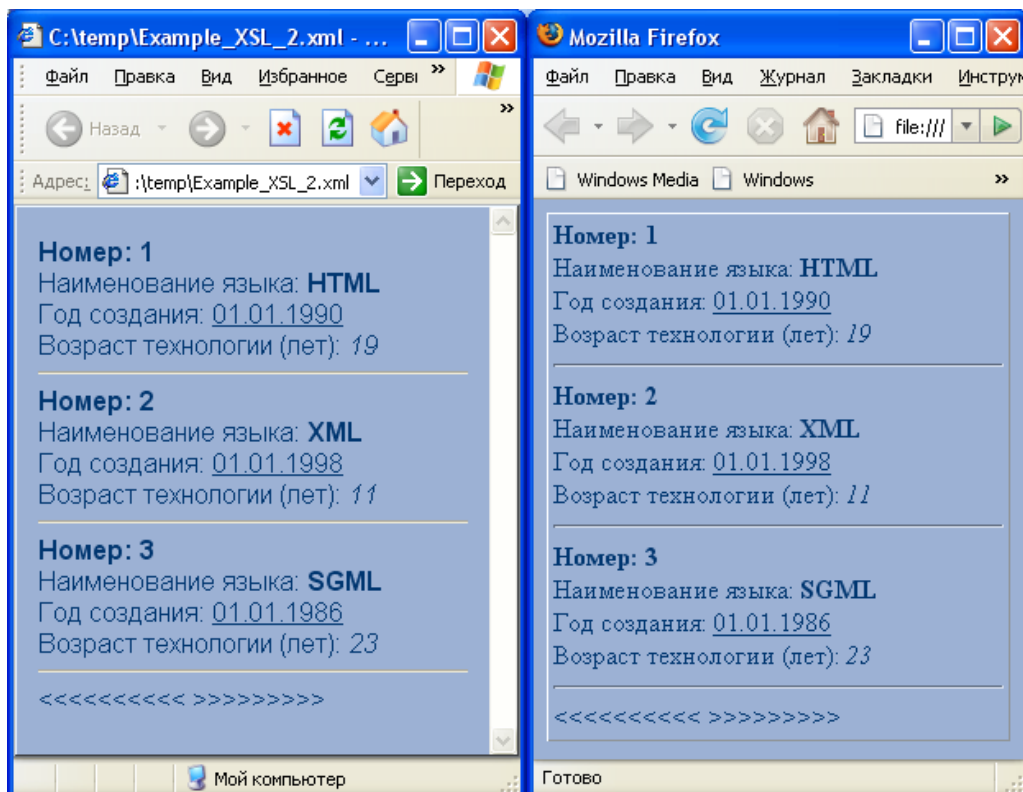
<!-- ++++++ -->

<!-- ++++++ -->
<!-- Шаблон обработки элемента year -->
<xsl:template match="year">
    Год создания: <U><xsl:value-of select="."/></U><BR/>
</xsl:template>
<!-- ++++++ -->

<!-- ++++++ -->
<!-- Шаблон обработки элемента howold -->
<xsl:template match="howold">
    Возраст технологии (лет): <I><xsl:value-of
select="."/></I><BR/>
</xsl:template>
<!-- ++++++ -->
</xsl:stylesheet>

```

### Просмотр результата в браузере:



XML файл в этом примере такой же, как и предыдущем случае. Для применения к нему XSLT-преобразования можно использовать инструкцию `<?xml-stylesheet type="text/xsl" href="название XSLT-преобразования.xsl" ?>`

В этом случае результат отображается в браузере. Или можно использовать отладчик XSLT, встроенный в XMLPad.

Рассмотрим более подробно текст преобразования XSLT.

```
<?xml version="1.0" encoding="Windows-1251" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<!-- ++++++ -->
<!-- Шаблон обработки корневого элемента XML - документа -->
<xsl:template match="/">
```

Как и в предыдущем примере, здесь присутствует шаблон для корневого элемента. Но в отличие от предыдущего примера здесь присутствуют и другие шаблоны.

```
<HTML>
<HEAD>
<LINK href="met.css" rel="stylesheet" type="text/css"/>
```

В секции HEAD документа HTML встречается тэг LINK, который связывает HTML-документ с внешней таблицей стилей. После обработки XML-документа с помощью XSLT к полученному HTML-документу будет применена таблица стилей CSS.

```
</HEAD>
<BODY>
<xsl:apply-templates/>
```

Главной инструкцией в этом примере является `apply-templates`. Эта инструкция выполняет следующие действия:

1. В текущем контексте (в текущем тэге) входного документа находит все непосредственно вложенные элементы.

2. Для каждого такого элемента пытаются найти соответствующий шаблон (template) и выполнить его. Соответствующим является шаблон вида `<xsl:template match="название элемента">`.

В нашем примере в корневой элемент непосредственно вложен элемент `languages`, поэтому далее будет вызван `<xsl:template match="languages">`.

Обратите внимание, что реальный корневой элемент документа «`languages`» считается вложенным в корневой элемент «`/`». Поэтому элементы `language` не вложены непосредственно в «`/`».

```
</BODY>
</HTML>
</xsl:template>
<!-- ++++++ -->

<!-- ++++++ -->
<!-- Шаблон обработки элемента languages -->
<xsl:template match="languages">
<!-- Вызов шаблона для элемента language (шаблон вызывается
необходимое количество раз) -->
  <xsl:apply-templates/>
```

В нашем примере в элемент `languages` непосредственно вложен элемент `language`, поэтому далее будут вызваны шаблоны `<xsl:template match="language">` для каждого элемента `language`.

```
</xsl:template>
<!-- ++++++ -->

<!-- ++++++ -->
<!-- Шаблон обработки элемента language -->
<xsl:template match="language">

<!-- Получение значения атрибута id (префикс @ означает атрибут) -->
  <B>Номер: <xsl:value-of select="@id"/></B><BR/>

<!-- Вызов шаблонов для элементов name, year и howold -->
  <xsl:apply-templates/>
```

В нашем примере в элемент `language` непосредственно вложены элементы `name`, `year` и `howold`. Инструкция `apply-templates` будет вызывать шаблоны для этих элементов.

```
<HR/>
</xsl:template>
<!-- ++++++ -->
<!-- ++++++ -->
<!-- Шаблон обработки элемента name -->
<xsl:template match="name">
    Наименование языка: <B><xsl:value-of select="."/></B><BR/>
```

Так как в этом шаблоне мы находимся в контексте элемента `name` (то есть уже «провалились» внутрь элемента `name`), то для получения значения текущего элемента надо использовать XPath-выражение «`.`» или «`self:*`».

Если вместо выражения `<xsl:value-of select="."/>` использовать `<xsl:value-of select="name"/>`, то оно вернет пустое значение, так как будет искать несуществующий элемент `name` внутри текущего элемента `name`.

Шаблоны для элементов `year` и `howold` построены аналогично шаблону для элемента `name`.

```
</xsl:template>
<!-- ++++++ -->
<!-- ++++++ -->
<!-- Шаблон обработки элемента year -->
<xsl:template match="year">
    Год создания: <U><xsl:value-of select="."/></U><BR/>
</xsl:template>
<!-- ++++++ -->
<!-- ++++++ -->
<!-- Шаблон обработки элемента howold -->
<xsl:template match="howold">
    Возраст технологии (лет): <I><xsl:value-of
select="."/></I><BR/>
```

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

```
<!-- ++++++ -->
```

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

Отметим, что текстовое содержимое элемента `CDATA_Example` скопировано в выходной поток, что в нашем случае является нежелательным эффектом. Поэтому, в преобразовании лучше создавать шаблоны для всех возможных элементов. Например, если добавить следующий шаблон,

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

то ненужные символы угловых скобок не будут отображаться. Этот шаблон фактически указывает, что для элемента `CDATA_Example` не нужно выполнять никаких действий.

Вместо этого можно использовать для элемента `apply-templates` атрибут `select`. Этот атрибут содержит XPath-выражение, которое указывает, для каких элементов нужно искать и выполнять шаблоны.

Если в шаблоне обработки корневого элемента команду

```
<xsl:apply-templates/>
```

заменить на

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

то элемент `CDATA_Example` не будет обрабатываться и ненужные символы угловых скобок не будут отображаться. Так как элемент `language` не вложен непосредственно в `</>`, то используется XPath-выражение `</language>`, а не `<language>`.

Можно или создавать шаблоны для всех элементов входного документа или указывать область видимости в атрибуте `select`.

В атрибуте `select` может быть указано произвольное XPath-выражение, которое не обязательно возвращает непосредственно вложенные элементы.

Таким образом, использование нескольких элементов `template` и конструкции `apply-templates` позволяет разрабатывать преобразования, адаптирующиеся к структуре входного документа.

## **2 Условия лабораторных работ**

### **2.1 Описание структур данных с использованием XML**

Разработайте пример описания выбранной Вами предметной области в виде документа XML. Документ должен содержать около 30-50 XML-элементов.

Разработанный документ XML должен содержать элементы описания структур данных в виде множества (или массива), дерева, графа.

### **2.2 Разработка XPath-запросов**

Для документа, разработанного в предыдущей лабораторной работе, разработайте запросы XPath, содержащие обращение к элементам и атрибутам, фильтры и сравнения.

### **2.3 Разработка XSLT-преобразования с фиксированной структурой**

Разработайте XSLT-преобразование, генерирующее выходной HTML-документ с фиксированной структурой.

Преобразование должно содержать один элемент `template`, соответствующий корневому элементу XML-документа.

### **2.4 Разработка XSLT-преобразования с адаптируемой структурой**

Разработайте XSLT-преобразование, генерирующее выходной HTML-документ, который адаптирован к структуре входного документа.

Преобразование должно содержать несколько элементов `template`, которые вызываются с использованием конструкции `apply-templates`.

## **3 Требования к отчетам**

Отчеты разрабатываются отдельно по каждой лабораторной работе. Отчет по каждой лабораторной работе должен включать:

- титульный лист;
- тексты XML-документов, XPath-запросов, XSLT-преобразований;
- результаты работы XPath-запросов, XSLT-преобразований.

## 4 Контрольные вопросы

1. Приведите примеры структур данных, которые можно описать с использованием XML.
2. Для чего предназначен язык XPath?
3. Какие основные выражения используются в XPath?
4. Чем отличается обращение к элементам от обращения к атрибутам в XPath?
5. Каким образом в XPath можно использовать фильтры и операторы сравнения?
6. Что такое контекстный XPath-запрос и чем он отличается от неконтекстного? Почему контекстные XPath-запросы так широко используются?
7. Для чего предназначена технология XSLT?
8. Какие три варианта преобразований обычно выполняют с помощью XSLT?
9. Что такое XSLT-процессор?
10. Какие элементы технологии XSLT позволяют создавать преобразования, адаптирующиеся к структуре входного документа?

## 5 Литература

1. Расширяемый язык разметки (XML) 1.0 (вторая редакция), 2000. [электронный ресурс] – Режим доступа: <http://www.rol.ru/news/it/helpdesk/xml01.htm> – Загл. с экрана.
2. Язык XML Path (XPath) версия 1.0, 1999. [электронный ресурс] – Режим доступа: <http://www.rol.ru/news/it/helpdesk/xpath01.htm> – Загл. с экрана.



3. Язык преобразований XSL (XSLT) версия 1.0, 1999. [электронный ресурс] – Режим доступа: <http://www.rol.ru/news/it/helpdesk/xslt01.htm> – Загл. с экрана.
4. Валиков А.Н. Технология XSLT. – СПб.: БХВ-Петербург, 2002. – 544 с.