

Московский государственный технический университет  
имени Н.Э. Баумана

---

Факультет «Информатика и системы управления»  
Кафедра «Системы обработки информации и управления»

Э.Н. Самохвалов, Г.И. Ревунков, Ю.Е. Гапанюк

# **ВВЕДЕНИЕ**

# **В XML-ТЕХНОЛОГИИ**

Электронное учебное издание

*Учебное пособие*  
*по дисциплине «XML – технологии»*  
*Второе издание*

Москва

(С) 2014 МГТУ им. Н.Э. БАУМАНА

УДК 004.6

*Рецензент: д.т.н., профессор Владимир Васильевич Сюзев*

*Рецензент: к.т.н. Геннадий Иванович Афанасьев*

**Самохвалов Э.Н., Ревунков Г.И., Гапанюк Ю.Е.**

Введение в XML – технологии. Электронное учебное издание. Второе изд. - М.: МГТУ имени Н.Э. Баумана, 2014. 568 с.

Издание содержит вопросы описания структур данных с использованием формата XML. Рассмотрены языки запросов XPath и XQuery, технологии XSLT и XForms, методы проверки структуры XML-документов с использованием DTD и XML-схем, основы работы с XML-ориентированной СУБД eXist, разработка веб-приложений на основе архитектуры XRX.

Для студентов МГТУ имени Н.Э. Баумана, обучающихся по специальности «Информатика и вычислительная техника».

*Рекомендовано НМС МГТУ им. Н.Э. Баумана*

*Электронное учебное издание*

**Самохвалов Эдуард Николаевич**

**Ревунков Георгий Иванович**

**Гапанюк Юрий Евгеньевич**

**ВВЕДЕНИЕ В XML-ТЕХНОЛОГИИ**

© 2014 МГТУ имени Н.Э. Баумана

## ОГЛАВЛЕНИЕ

<b>ПРЕДИСЛОВИЕ.....</b>	<b>9</b>
<b>ПРЕДИСЛОВИЕ КО ВТОРОМУ ИЗДАНИЮ.....</b>	<b>9</b>
<b>1 ЧАСТЬ 1. НАЧАЛЬНЫЕ СВЕДЕНИЯ О ТЕХНОЛОГИЯХ XML .....</b>	<b>10</b>
1.1 КРАТКАЯ ХАРАКТЕРИСТИКА ЯЗЫКА XML.....	10
1.2 СОДЕРЖИМОЕ XML-ДОКУМЕНТА .....	11
1.2.1 Элементы и атрибуты .....	12
1.2.2 Директивы анализатора.....	13
1.2.3 Секции CDATA .....	14
1.2.4 Комментарии.....	14
1.2.5 Пространства имен .....	14
1.3 ПРАВИЛЬНЫЕ И ДЕЙСТВИТЕЛЬНЫЕ XML-ДОКУМЕНТЫ.....	14
1.4 ОПИСАНИЕ СТРУКТУР ДАННЫХ С ПОМОЩЬЮ XML.....	16
1.5 ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ДЛЯ ИЗУЧЕНИЯ XML .....	23
1.6 ОСНОВНЫЕ ТЕХНОЛОГИИ, СВЯЗАННЫЕ С XML .....	26
1.7 КОНТРОЛЬНЫЕ ВОПРОСЫ.....	27
<b>2 ЧАСТЬ 2. ПРЕОБРАЗОВАНИЕ ДОКУМЕНТОВ XML.....</b>	<b>28</b>
2.1 ЯЗЫК XPATH 1.0 .....	28
2.1.1 Основные выражения.....	29
2.1.2 Фильтры и сравнения.....	33
2.1.3 Оси выборки .....	39
2.1.4 Функции XPath.....	45
2.1.5 Арифметические действия.....	48
2.1.6 Стандарты XPath 2.0 и XPath 3.0 .....	51
2.2 ТЕХНОЛОГИЯ XSLT 1.0.....	51
2.2.1 Отображение XML-документа с использованием CSS .....	51
2.2.2 Введение в XSLT .....	53
2.2.3 Пример преобразования XSLT.....	55
2.2.4 Использование отладчика XSLT в XMLPad.....	63
2.2.5 Пространства имен .....	65
2.2.6 Шаблоны с несколькими элементами <i>template</i> .....	69
2.2.7 Включение стилей.....	86
2.2.8 Импорт стилей.....	88
2.2.9 Использование сортировки.....	92
2.2.10 Формирование текстового документа .....	97
2.2.11 Копирование узлов .....	99
2.2.12 Расширения XSLT.....	101
2.2.13 Использование ключей поиска .....	105
2.2.14 Обработка документа из нескольких секций (использование переменных).....	109

2.2.15	Создание кросс-таблицы .....	113
2.3	ТЕХНОЛОГИЯ XSLT 2.0.....	121
2.3.1	Инструментальные средства для работы с XSLT 2.0 .....	121
2.3.2	Использование группировки .....	122
2.3.3	Пользовательские функции.....	128
2.4	ТЕХНОЛОГИЯ XSLT 3.0.....	132
2.5	МАТЕРИАЛЫ ДЛЯ ДАЛЬНЕЙШЕГО ИЗУЧЕНИЯ .....	133
2.6	КОНТРОЛЬНЫЕ ВОПРОСЫ.....	133
<b>3</b>	<b>ЧАСТЬ 3. ОПИСАНИЕ СТРУКТУРЫ ДОКУМЕНТОВ XML.....</b>	<b>135</b>
3.1	ИСПОЛЬЗОВАНИЕ DTD ДЛЯ ОПИСАНИЯ СТРУКТУРЫ ДОКУМЕНТОВ XML.....	135
3.1.1	Пример внешнего DTD .....	135
3.1.2	Пример несоответствия документа XML и DTD.....	141
3.1.3	Пример встроенного DTD.....	142
3.1.4	Графическое представление DTD.....	144
3.1.5	Генерация DTD и XML-схемы по XML-документу .....	146
3.1.6	Преобразование DTD в XML-схему и XML-схемы в DTD.....	147
3.2	ИСПОЛЬЗОВАНИЕ СХЕМ XML ДЛЯ ОПИСАНИЯ СТРУКТУРЫ ДОКУМЕНТОВ XML.....	147
3.2.1	Пример XML-схемы .....	147
3.2.2	Графическое представление схемы XML .....	152
3.2.3	Использование простых типов и ограничений .....	154
3.2.4	Списки и объединения.....	159
3.2.5	Простые элементы с атрибутами .....	163
3.2.6	Использование сложных (составных) типов.....	165
3.2.7	Использование групп элементов и атрибутов.....	176
3.2.8	Использование смешанной модели содержимого элемента.....	179
3.2.9	Использование «nil» для пустых элементов .....	181
3.2.10	Указание типа элемента в XML-документе.....	183
3.2.11	Использование аннотаций .....	184
3.2.12	Задание ключей и уникальности.....	186
3.2.13	Использование пространств имен.....	191
3.2.14	Создание XML-схем, состоящих из нескольких файлов .....	198
3.2.15	Шаблоны проектирования схем.....	203
3.2.16	Профиль XML-схем.....	211
3.2.17	Существующие стандарты описания XML-схем.....	213
3.3	МАТЕРИАЛЫ ДЛЯ ДАЛЬНЕЙШЕГО ИЗУЧЕНИЯ .....	214
3.4	КОНТРОЛЬНЫЕ ВОПРОСЫ.....	214
<b>4</b>	<b>ЧАСТЬ 4. ВВЕДЕНИЕ В XML–ОРИЕНТИРОВАННЫЕ СУБД И ЯЗЫК XQUERY .....</b>	<b>216</b>
4.1	ЕСТЕСТВЕННЫЕ И ПРИСПОСОБЛЕННЫЕ XML–ОРИЕНТИРОВАННЫЕ СУБД .....	216
4.2	ОСНОВЫ РАБОТЫ С СУБД «eXist».....	216
4.2.1	Установка eXist .....	217

4.2.2	Приборная панель eXist .....	217
4.2.3	Работа с пакетами .....	220
4.2.4	Среда разработки eXide.....	220
4.2.5	Работа с коллекциями .....	222
4.2.6	Поддержка WebDAV .....	224
4.3	ЯЗЫК XQUERY 1.0 .....	225
4.3.1	Подготовка и запуск примеров.....	225
4.3.2	Начальные сведения об XQuery .....	226
4.3.3	Модель данных .....	227
4.3.4	Выражения XPath 1.0.....	227
4.3.5	Последовательности.....	227
4.3.6	Арифметические выражения .....	228
4.3.7	Операторы сравнения .....	229
4.3.8	Создание элементов (конструкторы элементов) .....	230
4.3.9	Выражение FLWOR.....	232
4.3.10	Соединение документов с помощью выражения FLWOR .....	239
4.3.11	Условное выражение if-then-else .....	242
4.3.12	Выражения some и every .....	243
4.3.13	Действия над множествами .....	246
4.3.14	Сравнение узлов.....	248
4.3.15	Выражения ordered и unordered.....	249
4.3.16	Выражения для работы с типами данных .....	249
4.3.17	Обновление данных в базе данных (XQuery Update Facility).....	252
4.4	РАСШИРЕНИЯ XQUERY 3.0.....	253
4.4.1	Группировка.....	254
4.4.1	Оператор ветвления .....	255
4.4.2	Обработка исключений.....	255
4.4.3	Оператор map (!).....	256
4.5	ТЕХНОЛОГИЯ XQUERYX .....	256
4.6	СОЗДАНИЕ СЕРВЕРНЫХ СЦЕНАРИЕВ НА ЯЗЫКЕ XQUERY .....	259
4.6.1	Особенность использования кодировок файлов в eXist.....	259
4.6.2	Простой пример серверного сценария и модуля.....	260
4.6.3	Обработка документа из нескольких секций.....	266
4.6.4	Пример, реализующий простую CRUD-функциональность .....	270
4.7	XQUERY и XSLT .....	283
4.8	МАТЕРИАЛЫ ДЛЯ ДАЛЬНЕЙШЕГО ИЗУЧЕНИЯ .....	284
4.9	КОНТРОЛЬНЫЕ ВОПРОСЫ.....	284
<b>5</b>	<b>ЧАСТЬ 5. ТЕХНОЛОГИЯ XFORMS.....</b>	<b>286</b>
5.1	ОСНОВНЫЕ ОТЛИЧИЯ XFORMS-ФОРМ ОТ HTML-ФОРМ .....	286
5.2	ОСОБЕННОСТИ ТЕХНОЛОГИИ XFORMS.....	287
5.3	ПРОГРАММНЫЕ ПРОДУКТЫ ДЛЯ РАБОТЫ С XFORMS.....	288

5.3.1	<i>XForms-процессор «Orbeon Forms»</i> .....	288
5.3.2	<i>XForms-процессор «betterFORM»</i> .....	289
5.3.3	<i>XForms-процессор «XSLTForms»</i> .....	289
5.3.4	<i>XForms-процессор «Mozilla XForms»</i> .....	290
5.3.5	<i>Преобразование XML-схем в XForms-формы</i> .....	290
5.3.6	<i>Microsoft InfoPath</i> .....	290
5.4	РАБОТА С ПРИМЕРАМИ В ЭТОЙ ГЛАВЕ .....	291
5.5	БАЗОВЫЕ ПРИМЕРЫ XFORMS-ФОРМ .....	292
5.6	МОДЕЛЬ ДАННЫХ ФОРМЫ.....	298
5.6.1	<i>Задание ограничений на типы данных</i> .....	299
5.6.2	<i>Связь данных формы, ограничений и элементов управления формы</i> .....	302
5.6.3	<i>Типы данных в XForms</i> .....	307
5.6.4	<i>Задание ограничений на данные с помощью элемента <code>xforms:bind</code></i> .....	312
5.6.5	<i>Использование нескольких моделей и экземпляров</i> .....	319
5.7	ЭЛЕМЕНТЫ УПРАВЛЕНИЯ ФОРМЫ.....	324
5.7.1	<i>Элемент <code>input</code></i> .....	327
5.7.2	<i>Элемент <code>secret</code></i> .....	327
5.7.3	<i>Элемент <code>textarea</code></i> .....	329
5.7.4	<i>Элемент <code>output</code></i> .....	331
5.7.5	<i>Элемент <code>upload</code></i> .....	335
5.7.6	<i>Элемент <code>range</code></i> .....	338
5.7.7	<i>Элементы задания подсказок и вывода сообщений</i> .....	340
5.7.8	<i>Элемент <code>trigger</code></i> .....	344
5.7.9	<i>Элемент <code>submit</code></i> .....	346
5.7.10	<i>Элемент <code>select</code></i> .....	347
5.7.11	<i>Элемент <code>select1</code></i> .....	359
5.7.12	<i>Элемент <code>group</code></i> .....	366
5.7.13	<i>Элемент <code>switch</code></i> .....	369
5.7.14	<i>Элемент <code>repeat</code></i> .....	377
5.8	ЭЛЕМЕНТЫ ОБРАБОТКИ СОБЫТИЙ .....	392
5.8.1	<i>Элемент <code>action</code></i> .....	392
5.8.2	<i>Элемент <code>setvalue</code></i> .....	393
5.8.3	<i>Элемент <code>delete</code></i> .....	394
5.8.4	<i>Элемент <code>insert</code></i> .....	394
5.8.5	<i>Элемент <code>setindex</code></i> .....	395
5.8.6	<i>Элемент <code>setfocus</code></i> .....	395
5.8.7	<i>Элемент <code>dispatch</code></i> .....	395
5.8.8	<i>Элемент <code>refresh</code></i> .....	396
5.8.9	<i>Элемент <code>load</code></i> .....	396
5.8.10	<i>Элемент <code>send</code></i> .....	396
5.8.11	<i>Элемент <code>message</code></i> .....	396
5.8.12	<i>Примеры обработки событий</i> .....	397

5.9	СОХРАНЕНИЕ ДАННЫХ ФОРМЫ .....	409
5.10	ИСПОЛЬЗОВАНИЕ ТАБЛИЦ СТИЛЕЙ CSS В XFORMS-ФОРМАХ .....	417
5.10.1	Применение стилей CSS к XForms-форме.....	417
5.10.2	Применение стилей CSS к результирующему HTML-документу.....	418
5.11	МАТЕРИАЛЫ ДЛЯ ДАЛЬНЕЙШЕГО ИЗУЧЕНИЯ.....	418
5.12	КОНТРОЛЬНЫЕ ВОПРОСЫ .....	418
<b>6</b>	<b>ЧАСТЬ 6. ТЕХНОЛОГИИ XINCLUDE, XLINK, XPOINTER.....</b>	<b>420</b>
6.1	ТЕХНОЛОГИЯ XINCLUDE .....	420
6.2	ТЕХНОЛОГИЯ XLINK.....	420
6.3	ТЕХНОЛОГИЯ XPOINTER.....	420
6.4	ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИЙ XINCLUDE И XPOINTER В EXIST.....	421
6.4.1	Элемент <i>xi:include</i> .....	422
6.4.2	Элемент <i>xi:fallback</i> .....	423
6.4.3	Добавление результата работы серверного сценария .....	424
6.4.4	Добавление фрагмента документа с помощью XPointer .....	425
6.5	МАТЕРИАЛЫ ДЛЯ ДАЛЬНЕЙШЕГО ИЗУЧЕНИЯ .....	426
6.6	КОНТРОЛЬНЫЕ ВОПРОСЫ.....	426
<b>7</b>	<b>ЧАСТЬ 7. РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЙ НА ОСНОВЕ ТЕХНОЛОГИИ XRX .....</b>	<b>428</b>
7.1	АРХИТЕКТУРА «КЛАССИЧЕСКОГО» ВЕБ-ПРИЛОЖЕНИЯ .....	428
7.2	АРХИТЕКТУРА ВЕБ-ПРИЛОЖЕНИЯ НА ОСНОВЕ ТЕХНОЛОГИИ XRX.....	430
7.2.1	Обобщенная архитектура .....	430
7.2.2	Проверка введенных данных .....	433
7.2.3	Детализированная архитектура.....	434
7.2.4	Ограничения XRX-технологии .....	437
7.3	ПРИМЕР РАЗРАБОТКИ ВЕБ-ПРИЛОЖЕНИЯ НА ОСНОВЕ ТЕХНОЛОГИИ XRX.....	438
7.3.1	Постановка задачи.....	438
7.3.2	Логическая модель данных.....	438
7.3.2.1	Описание сущностей .....	438
7.3.2.2	Описание связей .....	439
7.3.2.3	Диаграмма сущность-связь.....	439
7.3.3	Физическая модель данных в СУБД «eXist» .....	440
7.3.4	Отображение логической модели данных в физическую модель данных .....	442
7.3.5	Разработка приложения.....	447
7.3.5.1	Соглашения по наименованию элементов XRX-приложения .....	447
7.3.5.2	Использование шаблона проектирования «модель-вид-контроллер» .....	448
7.3.5.3	Использование редактора XQuery-сценариев .....	449
7.3.5.4	Справочник «Тип процессора» .....	449
7.3.5.4.1	Сценарий list_processor_1.2.xql.....	450
7.3.5.4.2	Сценарий edit_processor_1.2.xql.....	458
7.3.5.4.3	Сценарий data_processor_1.2.xql.....	464
7.3.5.4.4	Взаимодействие между сценариями справочника «Тип процессора».....	472
7.3.5.5	Справочник «Компьютер».....	475

7.3.5.5.1	Сценарий list_computer_1.1.xql.....	475
7.3.5.5.2	Сценарий edit_computer_1.1.xql.....	479
7.3.5.5.3	Сценарий data_computer_1.1.xql.....	496
7.3.5.5.4	Взаимодействие между сценариями справочника «Компьютер».....	504
7.3.5.6	Модуль формирования отчета.....	507
7.3.5.6.1	Сценарий report.xql.....	509
7.3.5.6.2	XSLT-преобразование report.xsl.....	527
7.3.5.6.3	Взаимодействие между сценариями модуля .....	531
7.3.5.7	Другие сценарии.....	533
7.3.5.7.1	Сценарий index.xql.....	533
7.3.5.7.2	Сценарий initdb.xql .....	536
7.3.5.7.3	Таблица стилей style.css .....	539
7.3.5.7.4	Таблица стилей xforms.css.....	542
7.3.5.7.5	Модуль module.xqm.....	542
7.4	ДАЛЬНЕЙШЕЕ УЛУЧШЕНИЕ ПРИМЕРА.....	563
7.5	МАТЕРИАЛЫ ДЛЯ ДАЛЬНЕЙШЕГО ИЗУЧЕНИЯ .....	564
7.6	КОНТРОЛЬНЫЕ ВОПРОСЫ.....	564
<b>ИСТОЧНИКИ .....</b>		<b>565</b>



## **Предисловие**

Данное учебное пособие предназначено для ознакомления с основами XML-технологий.

В пособии рассматриваются технологии XSLT и XForms, языки запросов XPath и XQuery, проверка структуры XML-документов с использованием DTD и XML-схем, основы работы с XML-ориентированной СУБД eXist, разработка веб-приложений на основе архитектуры XRX.

Пособие использует принцип обучения на основе примеров. Все изучаемые технологии рассматриваются в виде последовательности примеров, архив с примерами предоставляется в виде приложения к пособию.

Все программные продукты, используемые в пособии, являются свободно распространяемыми.

## **Предисловие ко второму изданию**

Во втором издании пособия были актуализированы версии используемых программных продуктов, переработана часть примеров, добавлены новые примеры.

Также добавлены новые разделы по технологиям XSLT 2.0, XQuery 3.0.

# 1 Часть 1. Начальные сведения о технологиях XML

## 1.1 Краткая характеристика языка 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 за 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 Содержимое XML-документа

**Пример 1.1. Рассмотрим пример простого документа XML:**

```
<?xml version="1.0" encoding="Windows-1251"?>
<!-- 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>
  <empty attr1="1" attr2="текст" />
```

```

<CDATA_Example>
  <![CDATA[
    <<<<<<<<<  >>>>>>>>

  ]]>
</CDATA_Example>
</languages>

```

Содержимое XML- документа представляет собой набор:

1. элементов и атрибутов;
2. директив анализатора;
3. секций CDATA;
4. комментариев;
5. пространств имен.

### 1.2.1 Элементы и атрибуты

Элемент – это основная структурная единица XML- документа. Заклячая слово «HTML» в тэги <name> </name>, определяется непустой элемент, называемый <name>, содержимым которого является «HTML». В общем случае в качестве содержимого элементов могут выступать любые части XML- документа: текст, вложенные элементы документа, секции CDATA, комментарии.

В примере определены «**простые**» элементы name (текстового типа), year (типа дата) и howold (целочисленный).

Если элемент содержит вложенные элементы, то он называется «**составным**» или «**сложным**». Примерами составных элементов являются language, languages.

Любой непустой элемент должен состоять из начального, конечного тэгов и данных, заключенных между ними.

Плоская модель данных превращается с использованием элементов в иерархическую систему с множеством возможных связей между элементами.

В XML документе, как правило, определяется хотя бы один элемент, называемый корневым, и с него программы-анализаторы начинают просмотр документа. В приведенном примере этим элементом является languages.

[Оглавление](#)

В случае если элемент не имеет содержимого, т.е. нет данных, которые он должен определять, он называется пустым. Начальный и конечные тэги пустого элемента как бы объединяются в один, и надо обязательно ставить косую черту перед закрывающей угловой скобкой. Примером является элемент `<empty />`.

В HTML примерами пустых элементов являются `<br/>`, `<hr/>`, `<img/>`.

Если при определении элементов необходимо задать какие-либо параметры, уточняющие его характеристики, то имеется возможность использовать атрибуты элемента. Атрибут – это пара (название = «значение»), которую надо задавать при определении элемента в начальном тэге.

Примеры использования элементов с атрибутами:

```
<language id="1">
```

```
<empty attr1="1" attr2="текст" />
```

В пустых элементах нет содержимого, но атрибуты могут использоваться.

Использование одинарных или двойных кавычек при задании значения атрибута, в отличие от HTML, является обязательным.

Обратим внимание, что простые элементы и атрибуты могут иметь одинаковое содержимое. То есть можно использовать только простые элементы или только атрибуты.

В начале развития XML в Интернете активно обсуждалось, когда лучше использовать простые элементы, а когда атрибуты.

По результатам дискуссий общепринятым является следующий подход: в виде элементов задаются основные данные, характеризующие предметную область, а в виде атрибутов задаются данные, носящие вспомогательный, уточняющий характер.

### 1.2.2 Директивы анализатора

Инструкции, предназначенные для анализаторов языка (PI - Processing Instruction), описываются в XML документе при помощи специальных тэгов – `<?` и `?>`. Эти инструкции используются для управления процессом разбора

документа. Наиболее часто инструкции используются при определении типа документа (например, `<? xml version="1.0"?>`).

### 1.2.3 Секции CDATA

Чтобы задать область документа, которую при разборе анализатор будет рассматривать как простой текст, игнорируя любые инструкции, но, в отличие от комментариев, иметь возможность использовать их в приложении, необходимо использовать тэги `<![CDATA[ и ]]>`. Внутри этого блока можно помещать любую информацию, которая может понадобиться программе-клиенту для выполнения каких-либо действий. В область CDATA можно помещать, например, инструкции JavaScript или данные, содержащие тэговые скобки, как в примере.

### 1.2.4 Комментарии

Комментариями является любая область данных, заключенная между последовательностями символов `<!-- и -->`. Такие же комментарии используются в документах HTML.

### 1.2.5 Пространства имен

Пространства имен позволяют использовать различные языки (словари) на основе XML.

Например, если разработаны языки разметки физических и химических формул на основе XML, то оба они могут содержать тэг `formula`. Пространства имен позволяют «различать» физическую и химическую формулу, несмотря на одинаковое название тэгов.

Более подробно использование пространств имен рассмотрено в разделе, посвященном XSLT.

## 1.3 Правильные и действительные XML-документы

XML-документы должны удовлетворять следующим требованиям:

1. В заголовке документа помещается объявление XML, в котором указывается язык разметки документа, номер его версии и дополнительная информация. Возможно указание кодировки (например, encoding="Windows-1251").
2. Каждый открывающий тэг, определяющий некоторую область данных в документе, обязательно должен иметь парный закрывающий тэг. В отличие от HTML нельзя опускать закрывающие тэги.
3. Документ содержит единственный корневой элемент.
4. В XML учитывается регистр символов.
5. Все значения атрибутов, используемых в определении тэгов, должны быть заключены в одинарные апострофы или двойные кавычки.
6. Вложенность тэгов в XML строго контролируется, поэтому необходимо следить за порядком следования открывающих и закрывающих тэгов. Пересечение области действия тэгов не разрешается. Например, конструкция `<a> <b>...</a> </b>` будет некорректной с точки зрения XML, так как пересекаются области действия тэгов `<a>` и `<b>`.
7. Вся информация, располагающаяся между начальным и конечным тэгами, рассматривается в XML как данные, и поэтому учитываются все символы форматирования (т.е. пробелы, переводы строк, табуляции не игнорируются, как в HTML).

Если XML- документ не нарушает эти правила, то он называется **правильным** (well-formed, иногда встречается переводы «формально-правильный», «хорошо сформированный»), и все анализаторы, предназначенные для разбора XML- документов, смогут корректно с ним работать. Приведенный выше пример документа является правильным XML–документом. Более подробно эти правила рассмотрены в спецификации XML [XML, 2000].

Однако, кроме проверки на формальное соответствие грамматике языка (синтаксический контроль), в документе могут присутствовать средства контроля над содержанием документа (семантический контроль).

Для того чтобы обеспечить семантическую проверку XML-документов, необходимо использовать анализаторы, производящие такую проверку и называемые верифицирующими.

Существует два способа контроля правильности XML-документа: DTD-определения (Document Type Definition), которые остались от SGML, и схемы данных (XML Schema). В отличие от SGML, определение DTD-правил или схем в XML не является обязательным.

Разница между схемами и DTD состоит в том, что схемы данных сами являются XML-документами, в DTD используется собственный синтаксис.

Документ XML, который удовлетворяет своему DTD или схеме данных, называется **действительным** (valid).

Обратим внимание, что в большинстве случаев вполне достаточно, чтобы документ был только правильным, например, при использовании технологии XSLT. Действительные документы используются относительно редко, в тех случаях, когда необходимо проверять структуру документа.

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

**В примере 1.2 приведено описание структур данных с использованием 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>
```

[Оглавление](#)



</Множество>

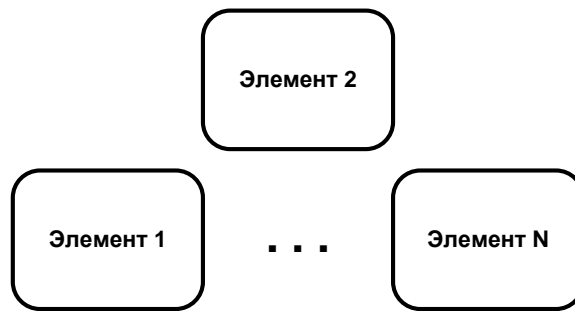


Рис. 1.1. Описание множества.

### 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>

```

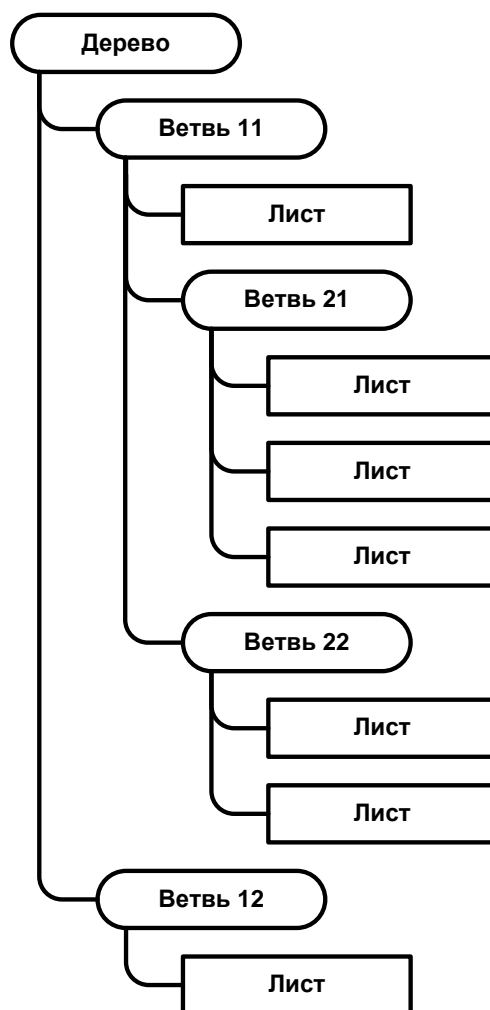


Рис. 1.2. Описание дерева.

Древовидные структуры являются «естественными» для модели данных XML.

### Реляционная таблица:

```
<?xml version="1.0" encoding="Windows-1251"?>
<!-- Таблица -->
<Таблица_1>
  <Запись>
    <Ключ>Значение ключевого поля</Ключ>
    <Поле_1>Значение поля 1</Поле_1>
    <Поле_2>Значение поля 2</Поле_2>
  </Запись>
  <!-- . . . -->
  <Запись>
    <Ключ>Значение ключевого поля</Ключ>
    <Поле_1>Значение поля 1</Поле_1>
    <Поле_2>Значение поля 2</Поле_2>
  </Запись>
</Таблица_1>
```

Таблица содержит последовательность записей. Поля записи могут быть представлены в виде элементов или атрибутов. Работать с реляционными данными в формате XML можно практически в любой современной технологии для работы с базами данных, например в ADO.NET.

При обмене реляционными данными в формате XML может возникнуть проблема, связанная с загрузкой первичных и вторичных ключей.

### Реляционные таблицы с ключами:

```
<?xml version="1.0" encoding="Windows-1251"?>
<База_данных>
  <Таблица_1>
    <Запись>
      <Первичный_ключ>1</Первичный_ключ>
      <Поле_1>Значение поля 1</Поле_1>
```

```

        <Поле_2>Значение поля 2</Поле_2>
    </Запись>
    <!-- . . . -->
</Таблица_1>

<Связанная_таблица>
    <Запись>
        <Первичный_ключ>1</Первичный_ключ>
        <Поле_3>Значение поля 3</Поле_3>
        <Вторичный_ключ_таблицы_1>1</Вторичный_ключ_таблицы_1>
    </Запись>
    <Запись>
        <Первичный_ключ>2</Первичный_ключ>
        <Поле_3>Значение поля 3</Поле_3>
        <Вторичный_ключ_таблицы_1>1</Вторичный_ключ_таблицы_1>
    </Запись>
    <!-- . . . -->
</Связанная_таблица>
</База_данных>

```

Такую структуру можно загрузить только в СУБД, которая поддерживает запись данных в ключевые поля (эту возможность поддерживают не все СУБД).

При сохранении связанных таблиц в формате XML их лучше сохранять в виде единого дерева без первичных и вторичных ключей.

### Реляционные таблицы без ключей:

```

<?xml version="1.0" encoding="Windows-1251"?>
<База_данных>
    <Таблица_1>
        <Запись>
            <Поле_1>Значение поля 1</Поле_1>
            <Поле_2>Значение поля 2</Поле_2>
            <Связанная_таблица>
                <Запись>
                    <Поле_3>Значение поля 3</Поле_3>

```

```

        </Запись>
        <Запись>
            <Поле_3>Значение поля 3</Поле_3>
        </Запись>
        <!-- . . . -->
    </Связанная_таблица>
</Запись>
<!-- . . . -->
</Таблица_1>
</База_данных>

```

При загрузке ключи необходимо генерировать «на лету».

При загрузке данных в MS SQL Server могут быть использованы «аннотированные схемы».

Аннотированные схемы представляют собой расширение XML-схем. В аннотированных схемах допускаются специальные аннотации, задающие их привязку к сущностям реляционной структуры: таблицам, полям, первичным и внешним ключам, отношениям между таблицами, благодаря чему данные, хранящиеся в MS SQL Server, можно запрашивать как XML-документ. Возможна также загрузка документа на основе аннотированной схемы. Более подробно эти возможности рассмотрены в [Ширшов, 2003].

### Граф (связи задаются в вершинах):

```

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

    <Вершина id="1">
        <Данные_o_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>

```

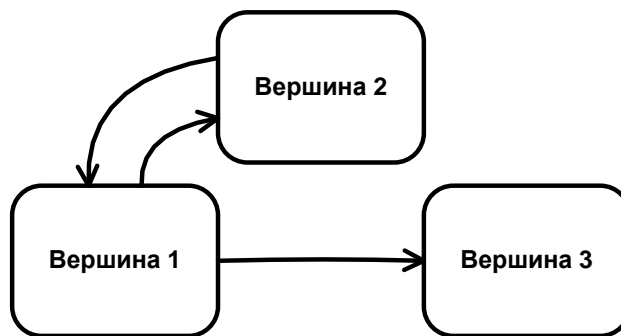


Рис. 1.3. Описание графа.

Вместо выходов можно использовать входы. Для неориентированных графов надо делать ссылки друг на друга из двух вершин.

Аналогичным образом можно описывать двудольные графы. В двудольном графе существует два класса вершин, и каждая вершина связана лишь с вершинами другого типа.

Можно использовать описание графа, в котором вершины и связи задаются отдельно друг от друга.

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

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

[Оглавление](#)

```

<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.

Для описания графов разрабатывается отдельная спецификация GraphML.

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

### **1.5 Инструментальные средства для изучения XML**

Одним из наиболее необходимых средств для изучения XML является XML-редактор. Наиболее известным коммерческим редактором XML является XML SPY.

Одним из лучших свободно распространяемых продуктов является XMLPad. Сайт XMLPad – «wmhelp.com». В этом редакторе реализованы наиболее необходимые функции XML SPY.

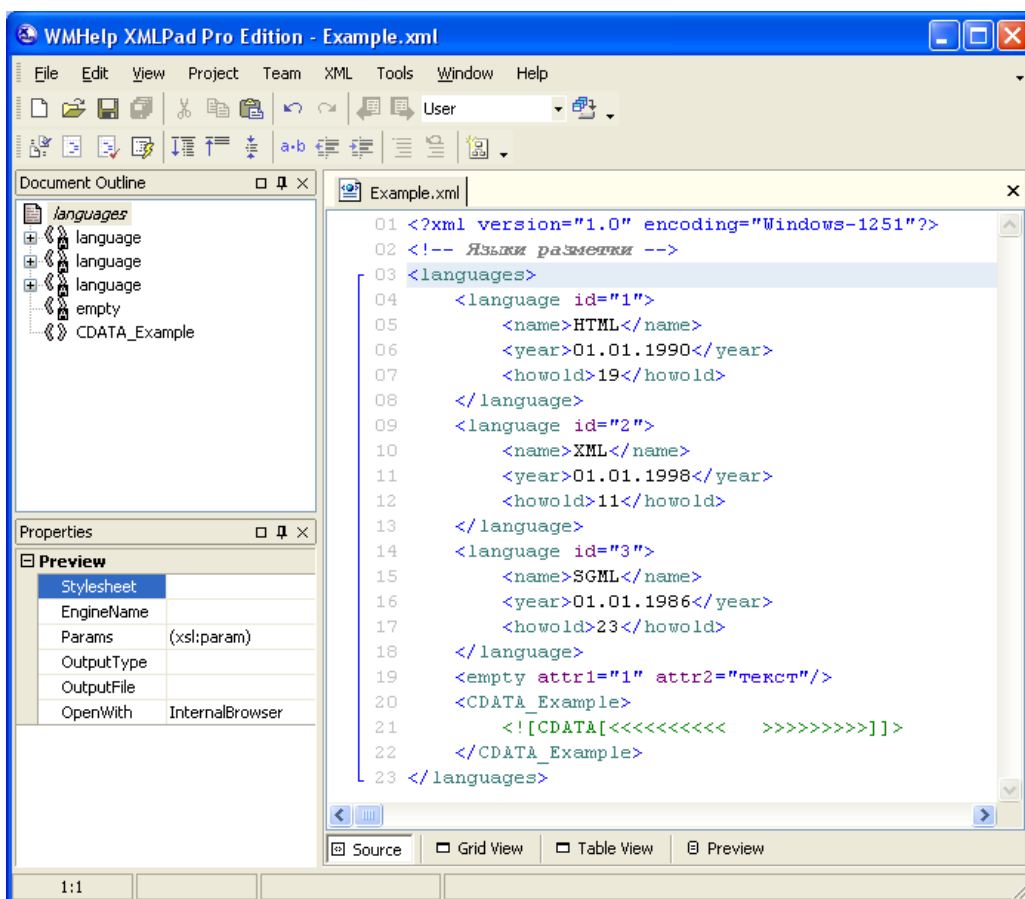
К таким функциям можно отнести:

1. Различные формы представления документов XML. Документ XML может быть представлен в редакторе в виде текста (Source), дерева

(Grid View), таблицы (Table View). Также документ может быть представлен в режиме предварительного просмотра Internet Explorer (Preview) без возможности редактирования.

2. Вычисление XPath-выражений. Выполнение XSLT-преобразований. Особенно полезной функцией является XSLT-отладчик.
3. Возможность генерации DTD и схем на основе документа XML. Возможность преобразования DTD и схем друг в друга. Графическое представление DTD и схем.

На следующем рисунке приведен пример представления документа в виде текста, дерева и таблицы.





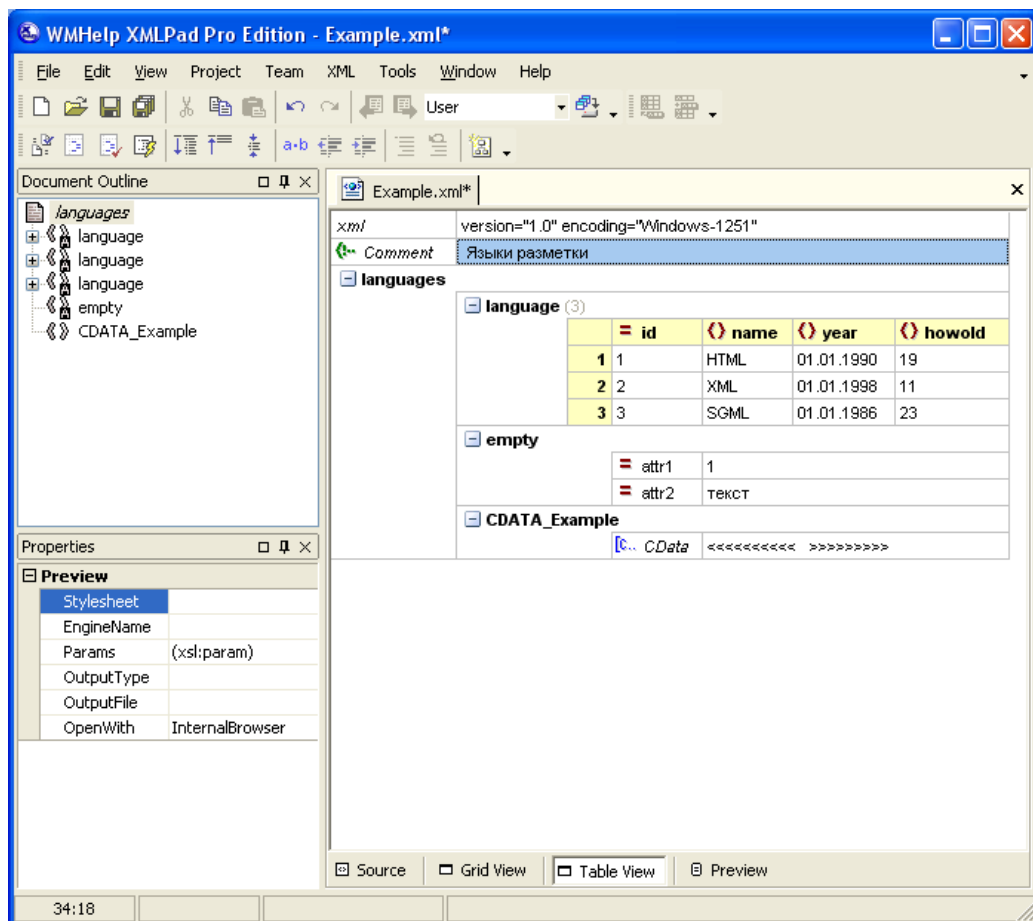
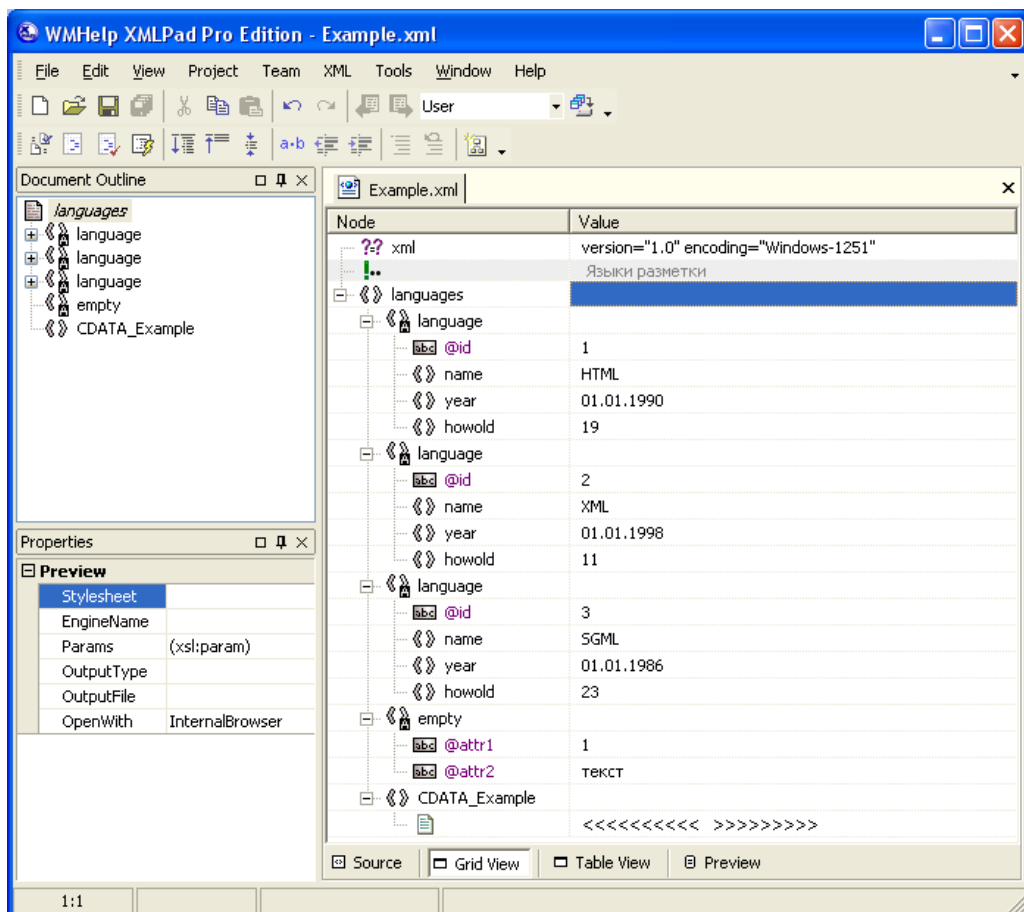


Рис. 1.4. Пример представления документа в виде текста, дерева и таблицы.

## Оглавление

## **1.6 Основные технологии, связанные с XML**

Перечислим основные технологии, которые предназначены для работы с XML, или в которых XML активно используется.

1. XML-парсеры, которые применяются для разбора документов XML на низком уровне. Различают такие разновидности парсеров, как SAX-парсеры и DOM-парсеры.

2. DTD-описания и схемы XML, которые предназначены для проверки структуры документа (проверка действительности документов XML). Далее эти технологии рассматриваются более подробно.

3. Язык XPath и технология XSLT, предназначенные для преобразования документов XML в HTML, XML или текстовый формат. Далее XPath и XSLT рассматриваются более подробно.

4. Технология XSL-FO, предназначенная для преобразования документов XML в печатное представление. В настоящее время вместо этой технологии часто применяется XSLT.

5. Хранение XML-документов в базе данных.

В настоящее время большинство современных СУБД позволяет работать с данными в формате XML, например Microsoft SQL Server, Oracle, Cache. Для работы с XML используются различные расширения языка SQL, иногда XQuery. В этих СУБД XML не является основным форматом хранения данных.

Существуют также специализированные XML-ориентированные СУБД, в которых XML является основным форматом хранения данных. Часто такие СУБД вообще не поддерживают реляционную модель. К таким СУБД можно отнести коммерческую Tamino, а также свободно распространяемые eXist и Sedna. В таких СУБД основным языком обработки данных, как правило, является XQuery. Далее более подробно рассматривается использование языка XQuery в СУБД eXist.

6. Ввод данных в формате XML. Основным стандартом является XForms. В некоторых продуктах применяются решения, не основанные на стандартах, например Microsoft InfoPath.

7. Использование XML для интеграции данных и приложений. Здесь можно выделить технологию веб-сервисов и связанные с ней стандарты SOAP, WSDL, UDDI, язык BPEL.

8. Стандарты семантического веба (RDF, RDFS, OWL, SPARQL). Большая часть стандартов семантического веба построена на использовании XML.

Существует также большое количество других технологий, стандартов и средств разработки, которые невозможно перечислить в рамках краткого обзора.

### **1.7 Контрольные вопросы**

1. Какие типы конструкций может включать XML-документ?
2. Что такое правильные и действительные документы?
3. Каким условиям должен удовлетворять документ XML, чтобы он считался правильным?
4. Какие структуры данных можно описать с использованием XML?
5. Какие технологии предназначены для работы с XML?

## 2 Часть 2. Преобразование документов XML

### 2.1 Язык XPath 1.0

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.

#### Пример 2.1. 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>
```

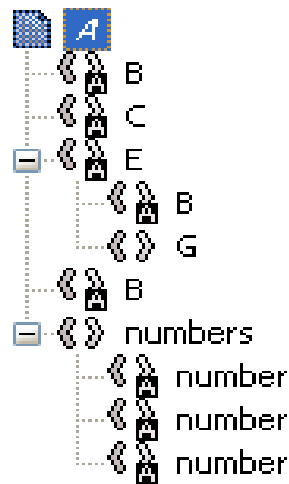


Рис. 2.1. Древоподобная структура документа.

Для выполнения XPath-запроса (XPath-выражения) в XMLPad необходимо выбрать пункт меню «XML/Evaluate XPath».

### 2.1.1 Основные выражения

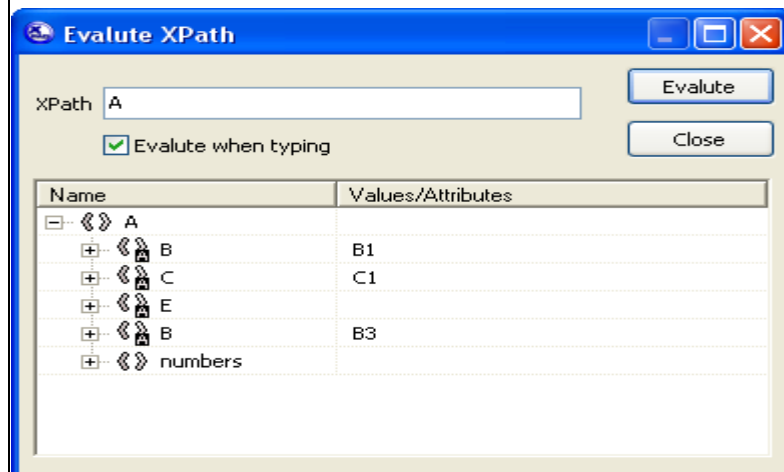
Примеры основных выражений XPath и результаты их выполнения приведены в следующей таблице.

Таблица 2.1. Основные выражения XPath.

XPath-выражение	Результат выполнения
/	<p>Корневой элемент.</p> <a href="#">Оглавление</a>

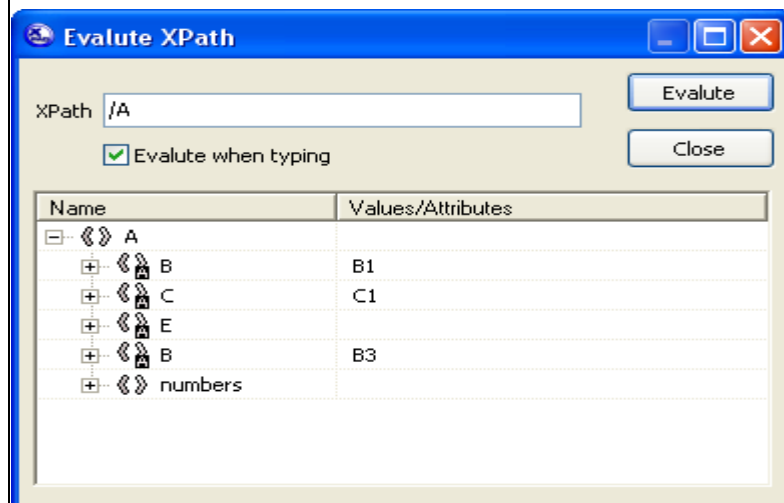
A

Элемент A.



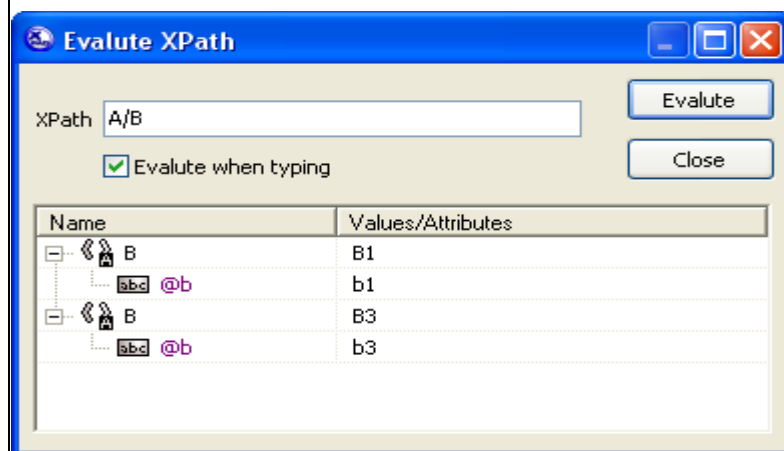
/A

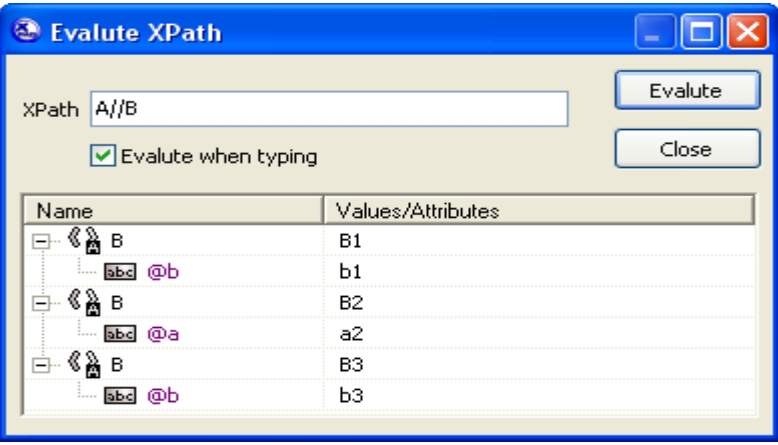
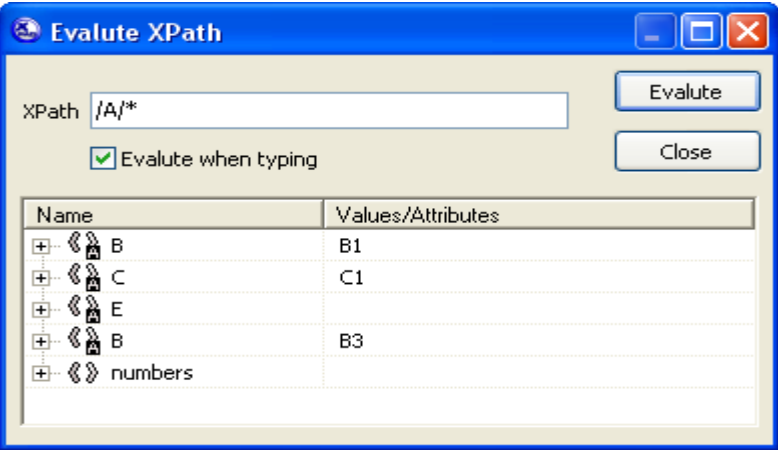
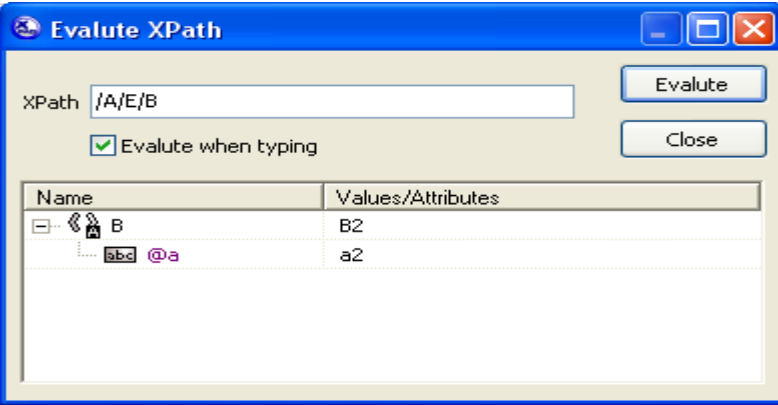
Элемент A, вложенный в корневой элемент. Результат выполнения такой же, как в предыдущем запросе.



A/B

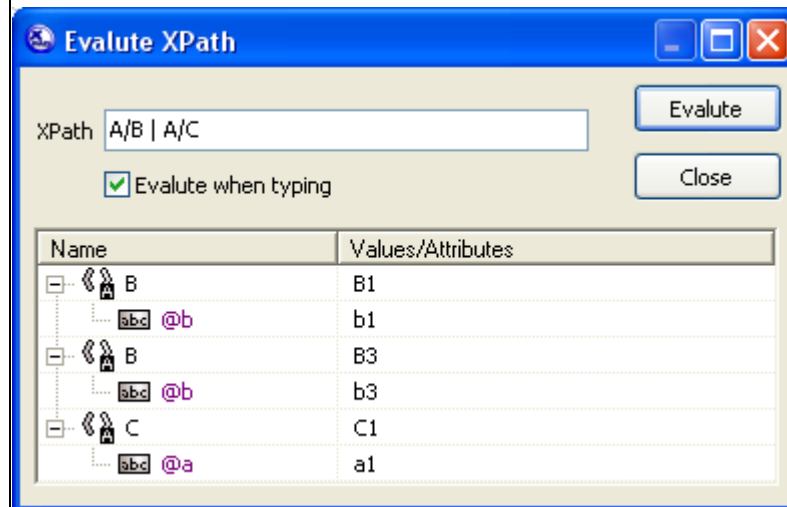
Элемент B, вложенный в элемент A.



A//B	<p>Элемент B, вложенный на любой глубине в элемент A. Обратите внимание, что элемент B со значением B2 не вложен непосредственно в элемент A.</p> 
/A/*	<p>Любой элемент, вложенный в A, вложенный в корневой элемент.</p> 
/A/E/B	<p>Элемент B, вложенный в E, вложенный в A, вложенный в корневой элемент.</p> 

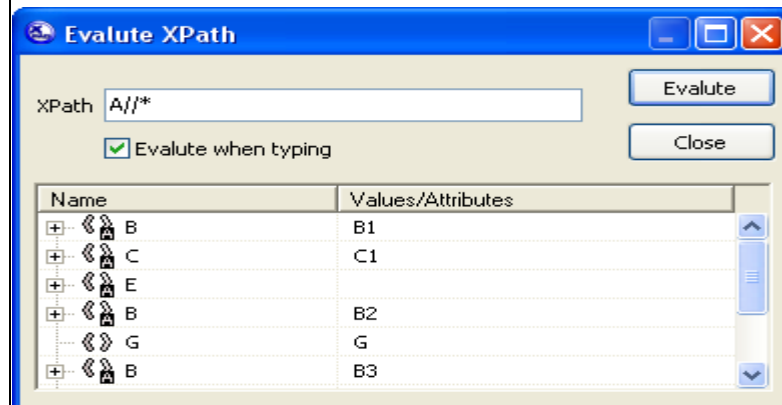
A/B | A/C

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



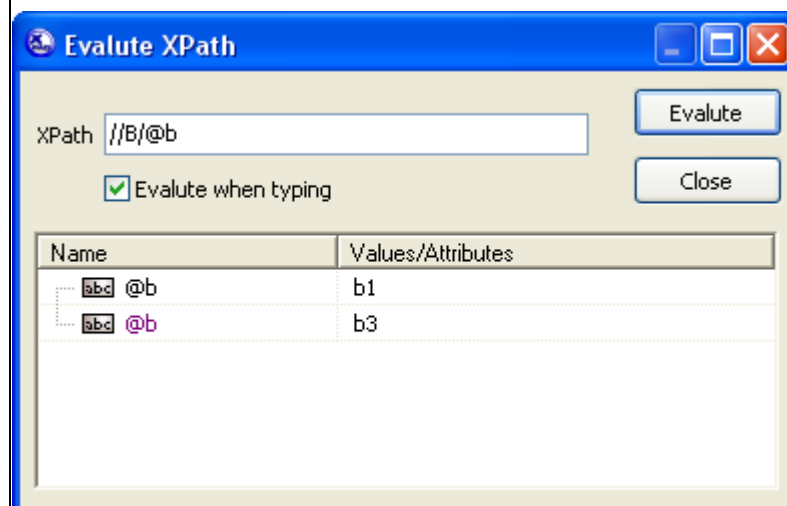
A//\*

Все элементы, вложенные в А на любой глубине.

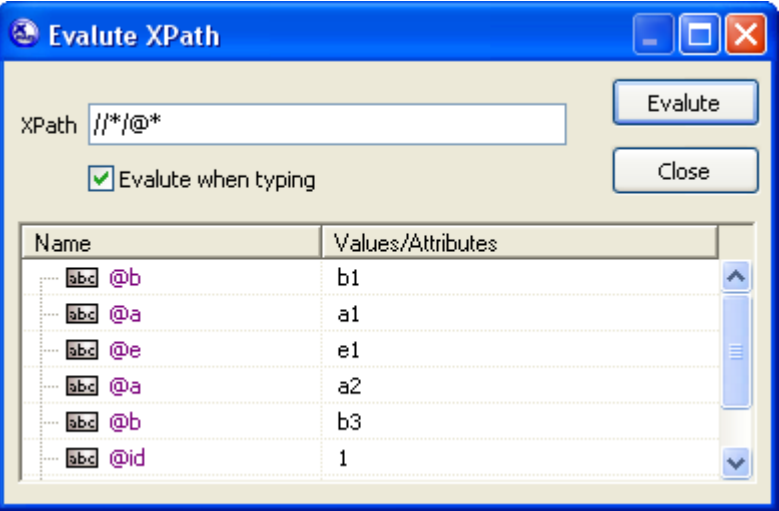


//B/@b

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





<p>//*/@*</p>	<p>Все атрибуты всех элементов документа.</p>  <table border="1" data-bbox="539 450 1273 712"> <thead> <tr> <th>Name</th> <th>Values/Attributes</th> </tr> </thead> <tbody> <tr> <td>abc @b</td> <td>b1</td> </tr> <tr> <td>abc @a</td> <td>a1</td> </tr> <tr> <td>abc @e</td> <td>e1</td> </tr> <tr> <td>abc @a</td> <td>a2</td> </tr> <tr> <td>abc @b</td> <td>b3</td> </tr> <tr> <td>abc @id</td> <td>1</td> </tr> </tbody> </table>	Name	Values/Attributes	abc @b	b1	abc @a	a1	abc @e	e1	abc @a	a2	abc @b	b3	abc @id	1
Name	Values/Attributes														
abc @b	b1														
abc @a	a1														
abc @e	e1														
abc @a	a2														
abc @b	b3														
abc @id	1														

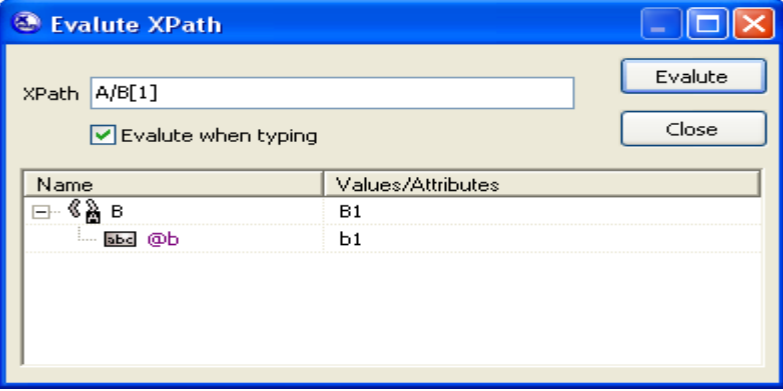
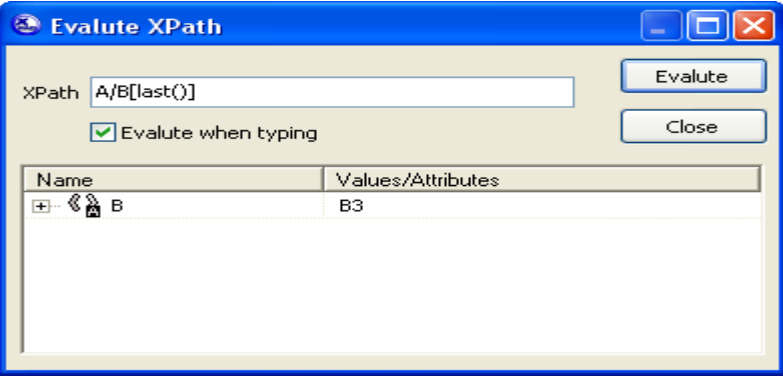
Обратите внимание, что если в запросе используется //, то такой запрос может «подтягивать» элементы на более высокий уровень иерархии. Например, в запросе A//\*, элементы E и G возвращаются на одном уровне иерархии, хотя в соответствии с примером элемент G вложен в E.

### 2.1.2 Фильтры и сравнения

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.

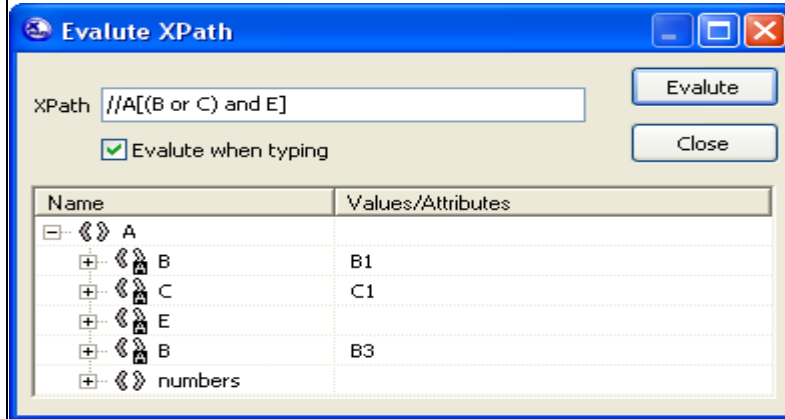
В качестве фильтра может быть задано число, которое определяет номер элемента. Это похоже на использование индексов в массиве. Или в качестве фильтра может быть задано условие поиска.

Таблица 2.2. Фильтры и сравнения.

A/B[1]	<p>Первый элемент B, вложенный в A.</p> <p>(В некоторых реализациях XPath нумерация начинается с 0, в некоторых с 1).</p> 
A/B[last()]	<p>Последний элемент B, вложенный в A.</p> <p>Функция last() возвращает индекс последнего элемента в последовательности.</p> 

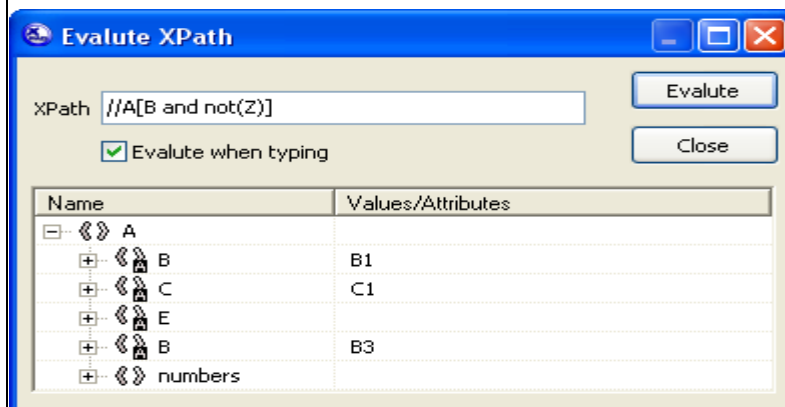
//A[(B or C) and E]

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.



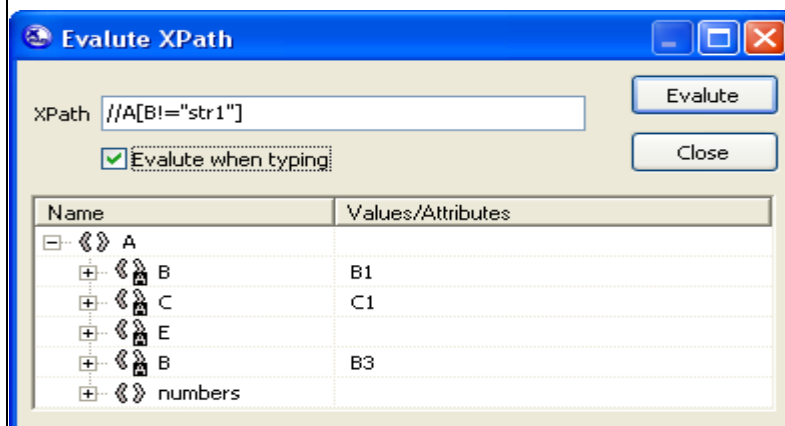
//A[B and not(Z)]

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



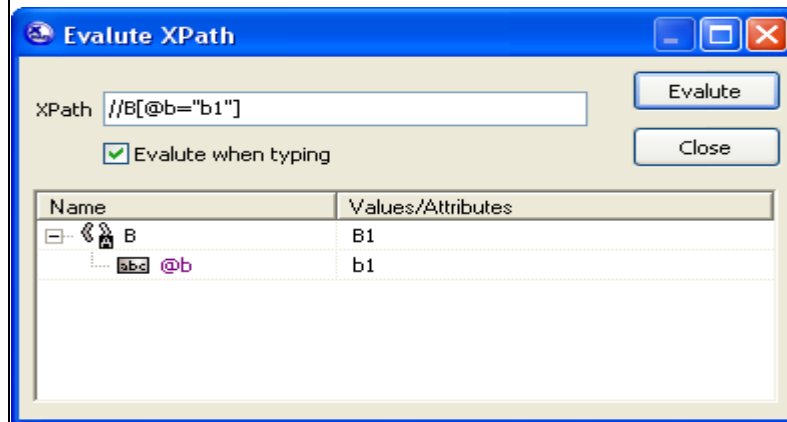
//A[B!="str1"]

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



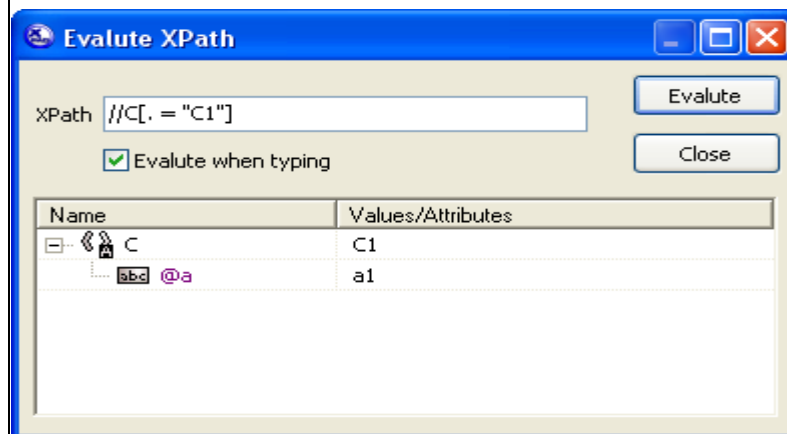
//B[@b="b1"]

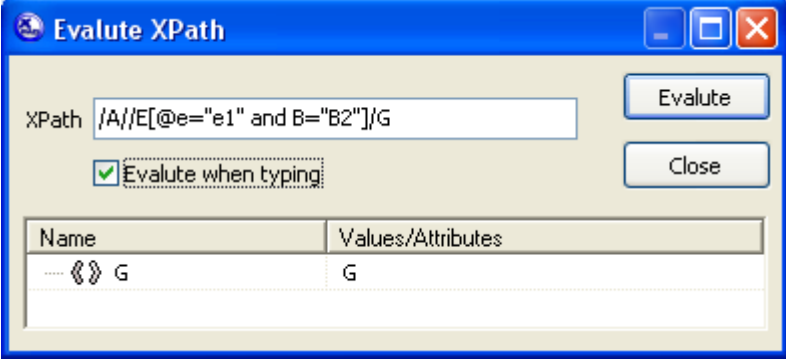
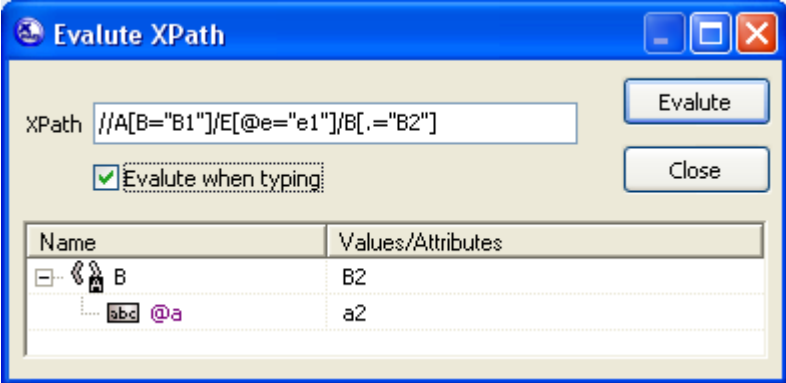
Поиск элемента B, у которого есть атрибут b=b1. Символ @ означает, что за ним следует название атрибута, а не элемента.

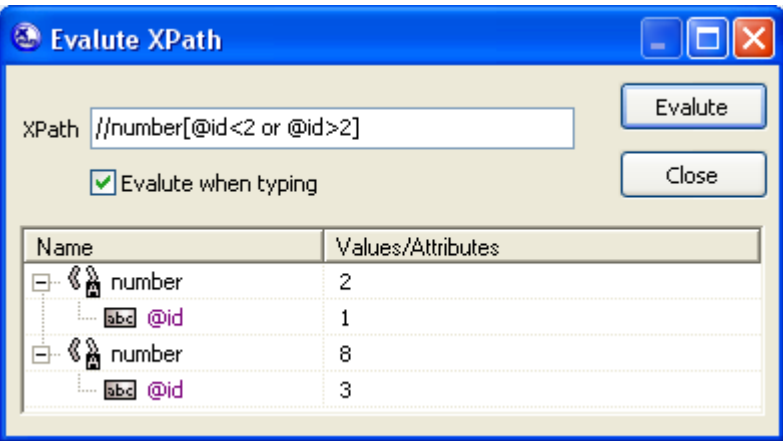
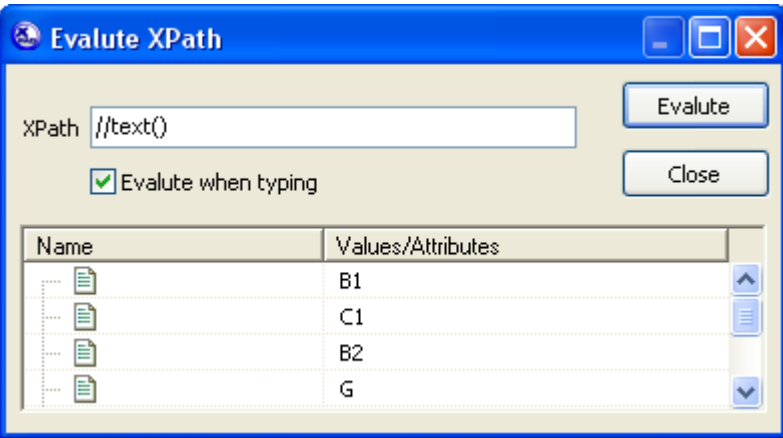


//C[. = "C1"]

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



<p><code>/A/E[@e="e1" and B="B2"]/G</code></p>	<p>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.</p> 
<p><code>//A[B="B1"]/E[@e="e1"]/B[.="B2"]</code></p>	<p>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.</p> <p>Этот пример показывает, что фильтры могут применяться к элементам на любой глубине вложенности и фильтрами могут быть снабжены несколько элементов в запросе.</p> 

<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.

### 2.1.3 Оси выборки

Оси выборки служат для определения направления движения по дереву узлов.

Таблица 2.3. Основные оси выборки.

Название	Описание	Сокращенная форма записи оси выборки
self	Текущий узел	.
child	Непосредственно вложенные узлы	/
parent	Родительский узел	..
descendant	Потомки узла на любой глубине вложенности	//
descendant-or-self	Узел и его потомки	
ancestor	Предки узла	
ancestor-or-self	Сам узел и его предки	
following	Все узлы после данного	
following-sibling	Все узлы этого же уровня после данного	
preceding	Все узлы перед данным	
preceding-sibling	Все узлы этого же уровня перед данным	
attribute	Узлы атрибутов	@
namespace	Узлы пространства имен	

Схематичное представление осей выборки в документе показано на рис. 2.2. Черным цветом показана древовидная структура документа, другими цветами оси

выборки. Фиолетовым цветом показан текущий элемент и соответствующая ему ось выборки «self». Зеленым цветом показаны оси выборки элементов вверх по дереву, синим цветом оси выборки элементов вниз по дереву. Оранжевым цветом показаны оси выборки элементов, которые следуют перед текущим элементом в документе, красным цветом показаны оси выборки элементов, которые следуют после текущего элемента в документе.

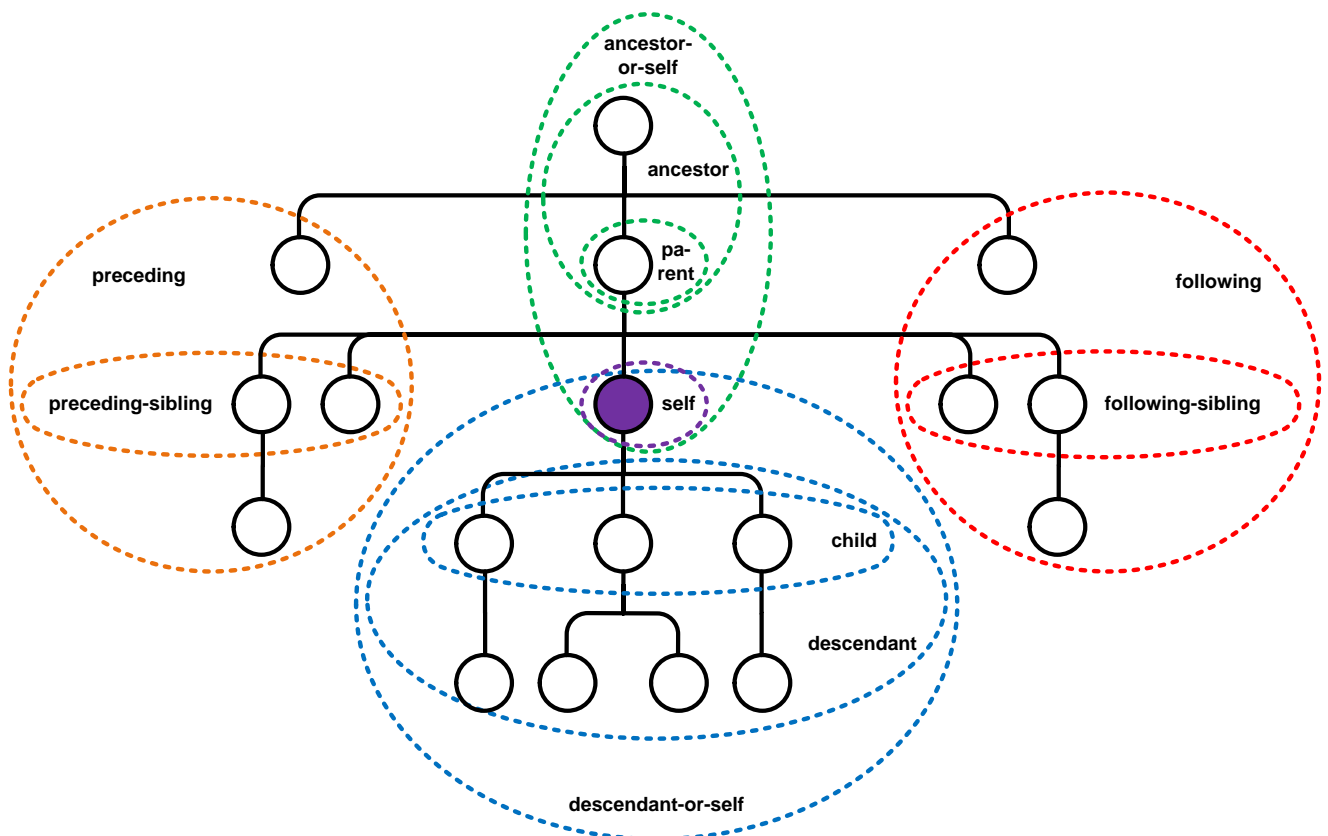
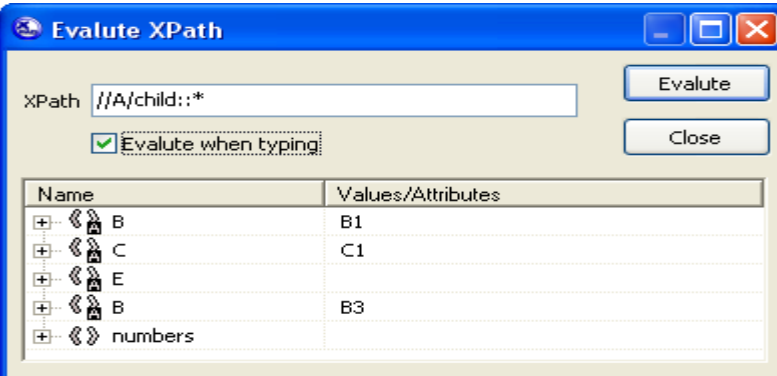
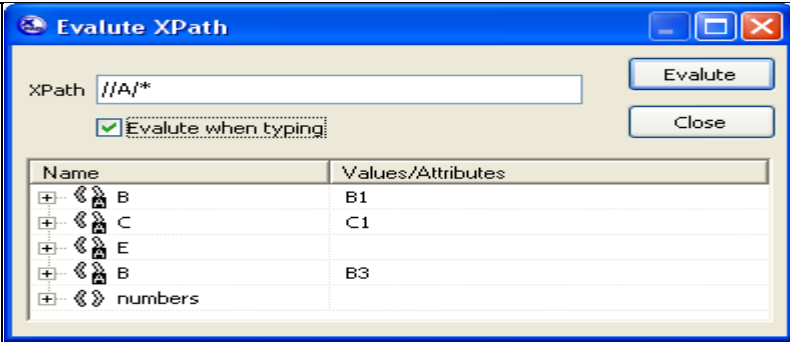
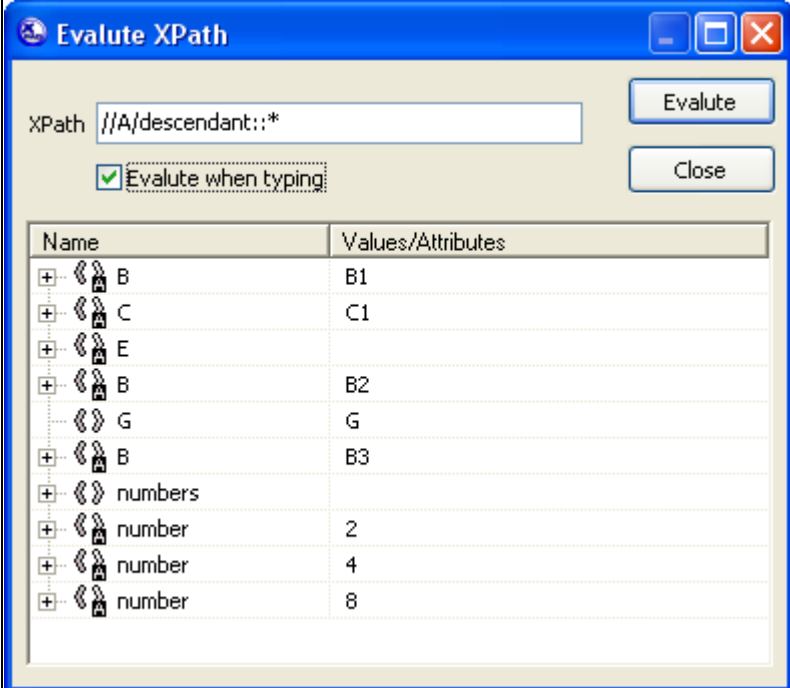
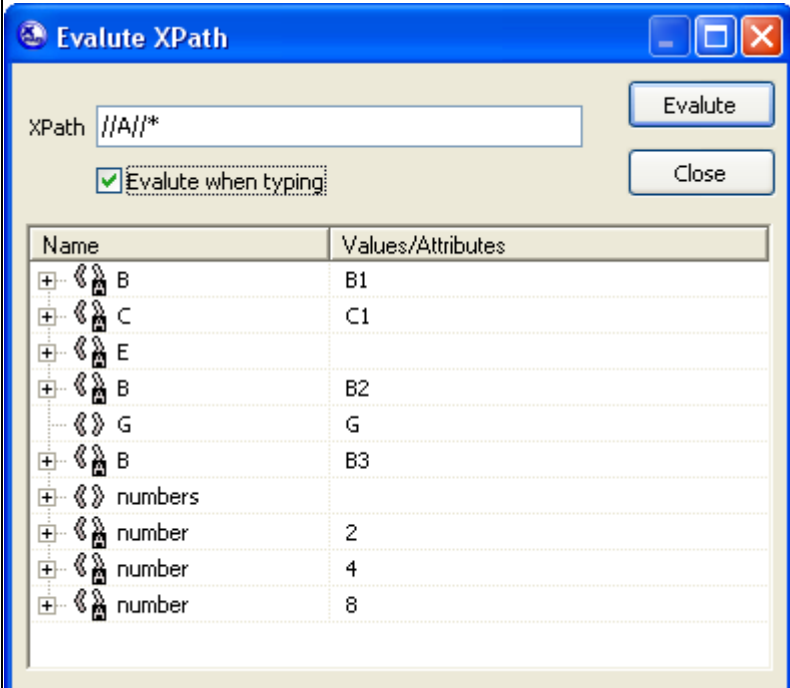


Рис. 2.2. Схематичное представление осей выборки.

Таблица 2.4. Примеры использования осей выборки.

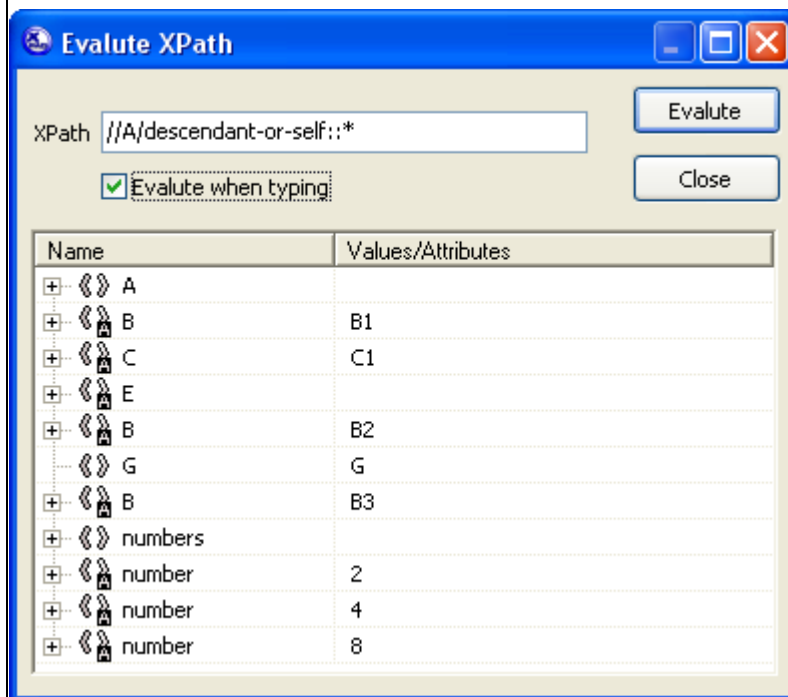
<p>//A/child::* или //A/*</p>	<p>Элементы, непосредственно вложенные в А.</p>  <table border="1"> <thead> <tr> <th>Name</th><th>Values/Attributes</th></tr> </thead> <tbody> <tr> <td>B</td><td>B1</td></tr> <tr> <td>C</td><td>C1</td></tr> <tr> <td>E</td><td></td></tr> <tr> <td>B</td><td>B3</td></tr> <tr> <td>numbers</td><td></td></tr> </tbody> </table>	Name	Values/Attributes	B	B1	C	C1	E		B	B3	numbers	
Name	Values/Attributes												
B	B1												
C	C1												
E													
B	B3												
numbers													



	 <p> <b>Evaluate XPath</b>          XPath: <input type="text" value="//A/*"/>  <input checked="" type="checkbox"/> Evaluate when typing          Evaluate Close       </p> <table border="1"> <thead> <tr> <th>Name</th><th>Values/Attributes</th></tr> </thead> <tbody> <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	B	B1	C	C1	E		B	B3	numbers											
Name	Values/Attributes																						
B	B1																						
C	C1																						
E																							
B	B3																						
numbers																							
//A/descendant::* или //A/**	<p>Элементы, вложенные в A на любой глубине.</p>  <p> <b>Evaluate XPath</b>          XPath: <input type="text" value="//A/descendant::*"/>  <input checked="" type="checkbox"/> Evaluate when typing          Evaluate Close       </p> <table border="1"> <thead> <tr> <th>Name</th><th>Values/Attributes</th></tr> </thead> <tbody> <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>B2</td></tr> <tr><td>G</td><td>G</td></tr> <tr><td>B</td><td>B3</td></tr> <tr><td>numbers</td><td></td></tr> <tr><td>number</td><td>2</td></tr> <tr><td>number</td><td>4</td></tr> <tr><td>number</td><td>8</td></tr> </tbody> </table>	Name	Values/Attributes	B	B1	C	C1	E		B	B2	G	G	B	B3	numbers		number	2	number	4	number	8
Name	Values/Attributes																						
B	B1																						
C	C1																						
E																							
B	B2																						
G	G																						
B	B3																						
numbers																							
number	2																						
number	4																						
number	8																						
	 <p> <b>Evaluate XPath</b>          XPath: <input type="text" value="//A/**"/>  <input checked="" type="checkbox"/> Evaluate when typing          Evaluate Close       </p> <table border="1"> <thead> <tr> <th>Name</th><th>Values/Attributes</th></tr> </thead> <tbody> <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>B2</td></tr> <tr><td>G</td><td>G</td></tr> <tr><td>B</td><td>B3</td></tr> <tr><td>numbers</td><td></td></tr> <tr><td>number</td><td>2</td></tr> <tr><td>number</td><td>4</td></tr> <tr><td>number</td><td>8</td></tr> </tbody> </table>	Name	Values/Attributes	B	B1	C	C1	E		B	B2	G	G	B	B3	numbers		number	2	number	4	number	8
Name	Values/Attributes																						
B	B1																						
C	C1																						
E																							
B	B2																						
G	G																						
B	B3																						
numbers																							
number	2																						
number	4																						
number	8																						

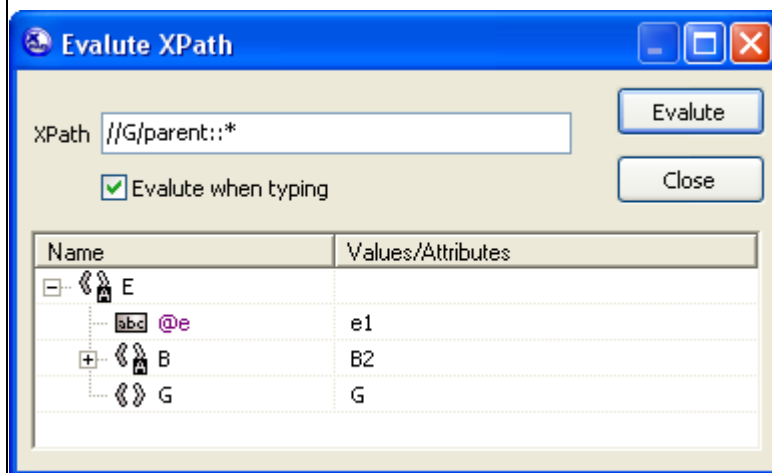
//A/descendant-or-self::\*

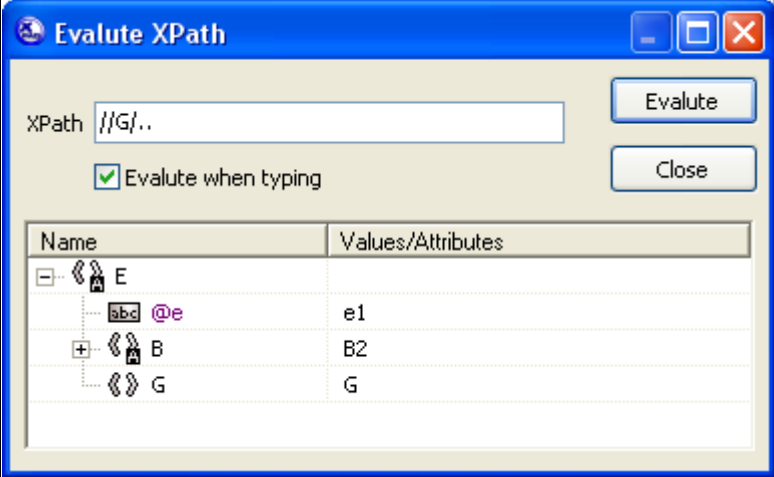
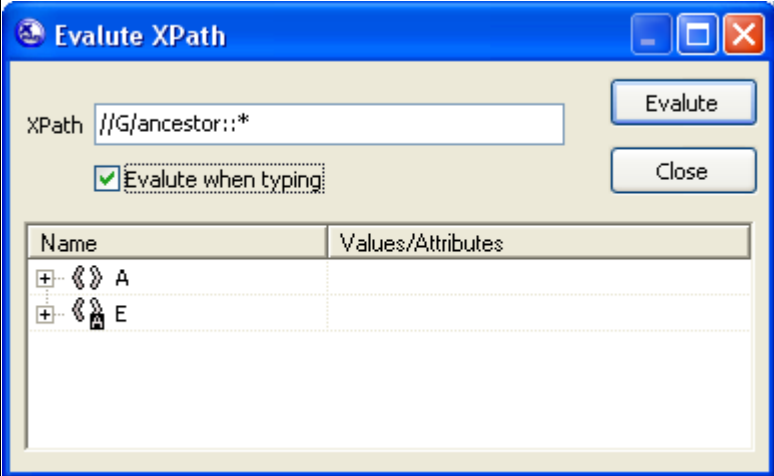
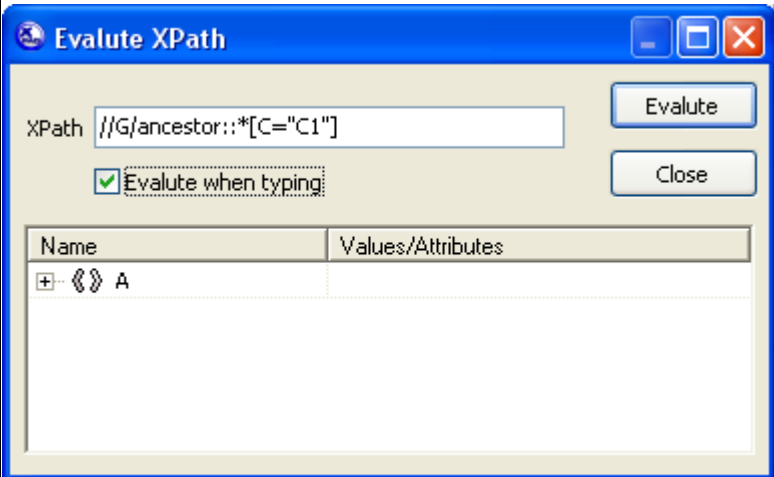
Элементы, вложенные в A на любой глубине и сам элемент A.



//G/parent::\* или //G/..

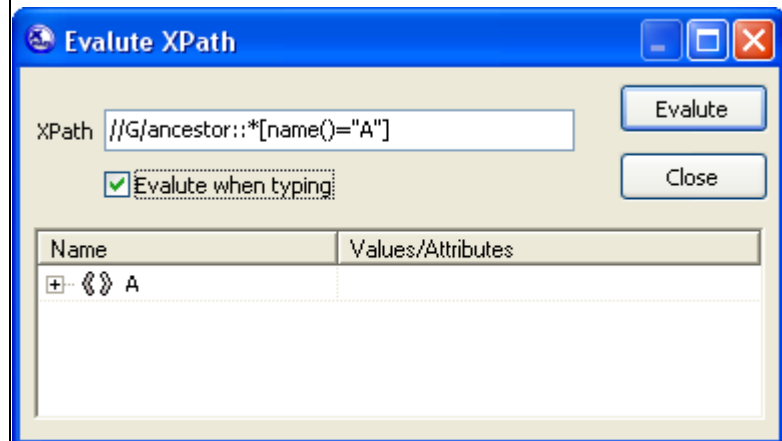
Узел, в который непосредственно вложен элемент G.



	
//G/ancestor::*	<p>Все узлы, в которые вложен элемент G.</p> 
//G/ancestor::*[C="C1"]	<p>Из узлов, в которые вложен элемент G, выбрать тот, у которого есть вложенный элемент C=C1</p> 

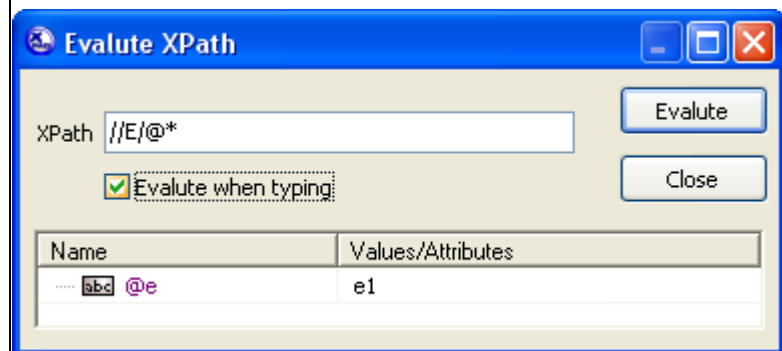
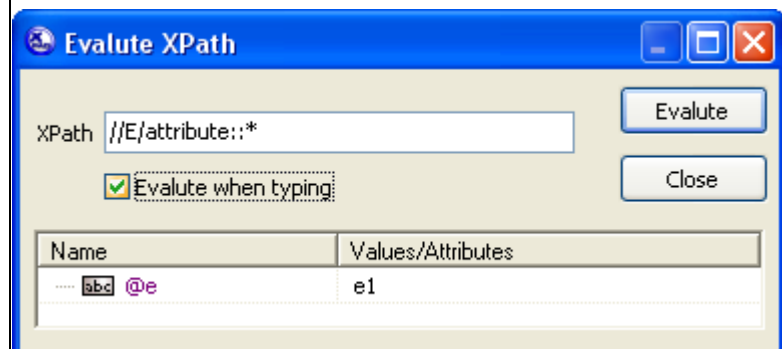
//G/ancestor::\*[name()='A']

Из узлов, в которые вложен элемент G, выбрать тот, который называется A.



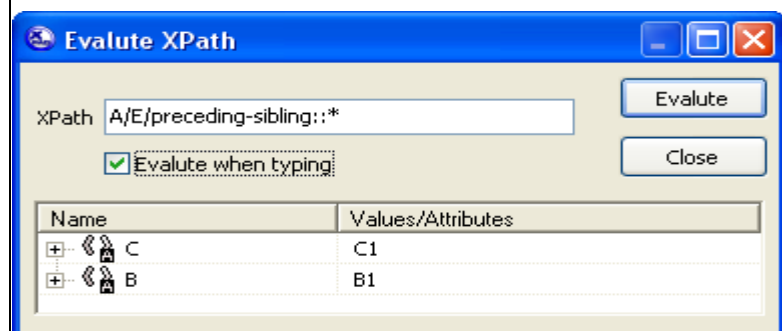
//E/attribute::\* или //E/@\*

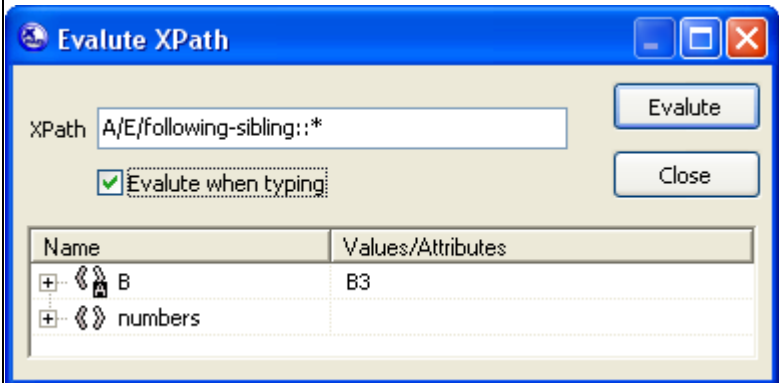
Все атрибуты элемента E.



A/E/preceding-sibling::\*

Узлы, следующие перед E на том же уровне иерархии.



A/E/following-sibling::*	<p>Узлы, следующие после E на том же уровне иерархии.</p> 
--------------------------	--

С точки зрения начального элемента (начальной точки) выполнения запроса можно выделить два вида запросов:

1. Запросы, в которых начальной точкой является корневой элемент документа (начинаются с « / »).
2. Контекстные запросы, в которых начальной точкой запроса является определенный контекст, то есть определенный элемент в документе.

В технологии XSLT обычно применяются контекстные запросы.

Использование осей выборки особенно полезно в контекстных запросах. Например, находясь в определенном контексте, мы не знаем, какой узел является родительским, но его можно определить с помощью оси parent.

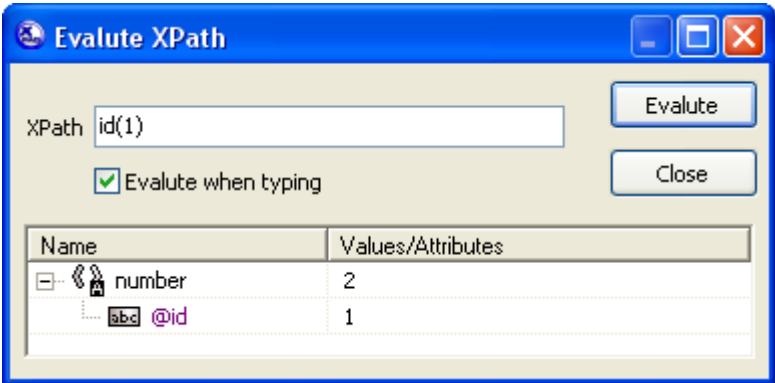
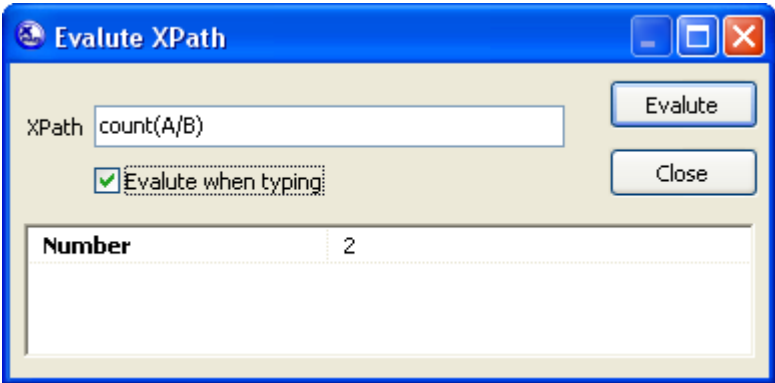
#### 2.1.4 Функции XPath

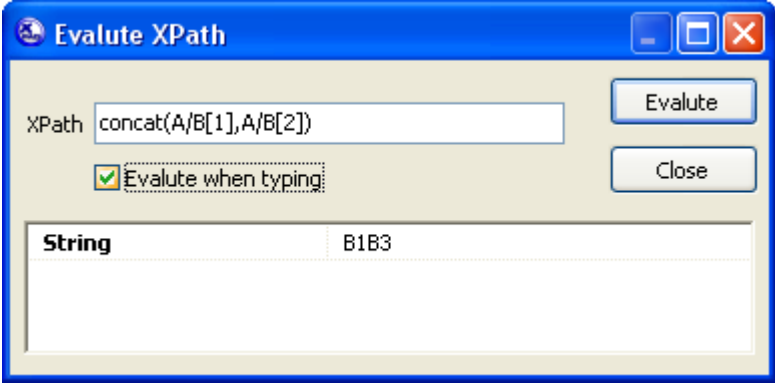
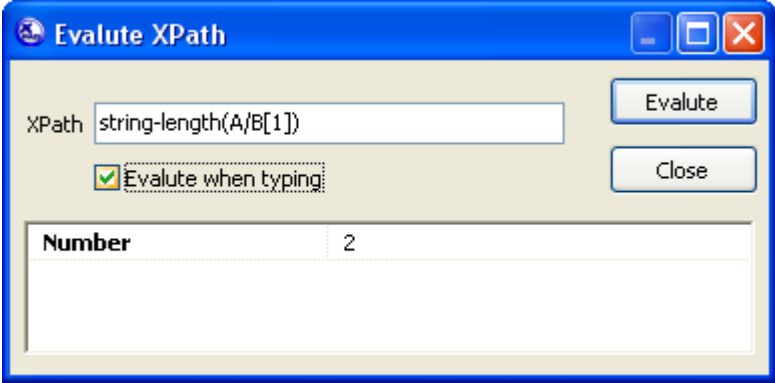
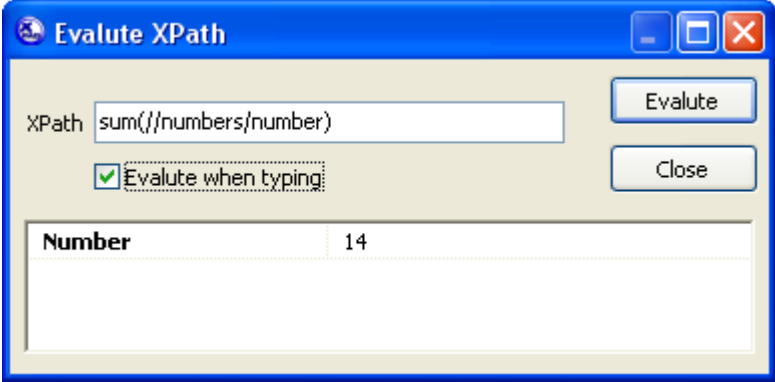
Для удобства работы с данными в XPath включена библиотека функций. Функции XPath можно разделить на четыре группы:

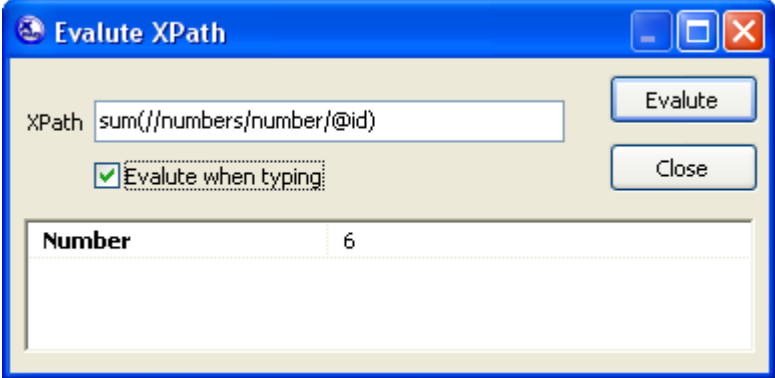
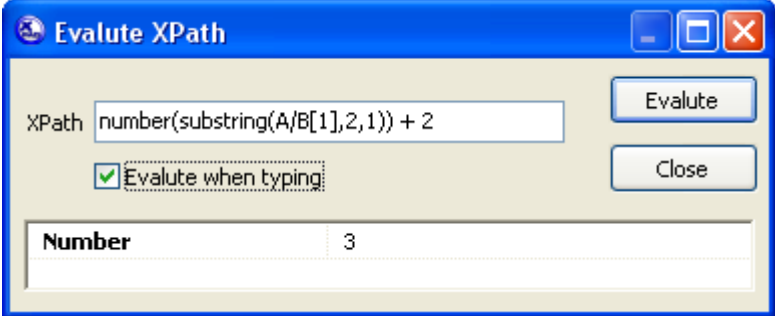
1. Функции для работы с элементами и атрибутами (например, такие функции, как last(возвращает индекс последнего элемента в последовательности), name(возвращает название элемента), id(поиск элемента по атрибуту id), count(определение количества элементов) ).
2. Функции для работы со строками (например, concat(соединение строк), substring(возвращает подстроку), string-length(длина строки) )

3. Функции для работы с булевыми значениями (например, функция `not`(инверсия значения)).
4. Функции для работы с числами (например, функция `sum`(сумма элементов), `number`(преобразование в число), функции `floor`, `ceiling`, `round` для округления).

Таблица 2.5. Примеры использования функций XPath.

id(1)	<p>Значение элемента, у которого атрибут id=1. Предполагается, что значение атрибута id всегда уникально в рамках документа.</p> 
count(A/B)	<p>Количество элементов B, вложенных в A.</p> 

concat(A/B[1],A/B[2])	<p>Соединение строк, хранящихся в A/B[1] и в A/B[2].</p> 
string-length(A/B[1])	<p>Длина строки, хранящейся в A/B[1].</p> 
sum(//numbers/number)	<p>Сумма элементов number, вложенных в numbers.</p> 

<code>sum(//numbers/number/@id)</code>	<p>Сумма атрибутов id элементов number, вложенных в numbers.</p> 
<code>number(substring(A/B[1],2,1)) + 2</code>	<p>Получение одного символа со второй позиции из значения элемента A/B[1] (это символ «1»), преобразование в число, прибавление 2.</p> 

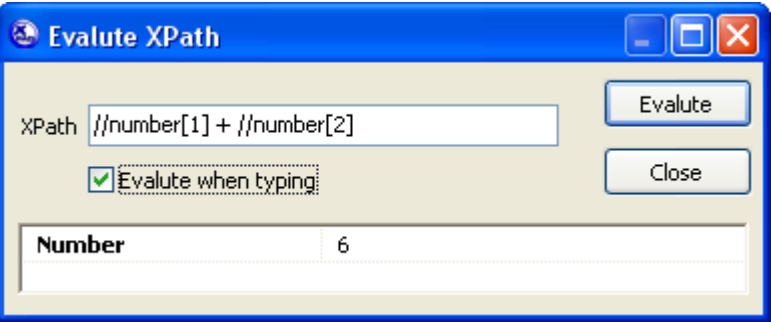
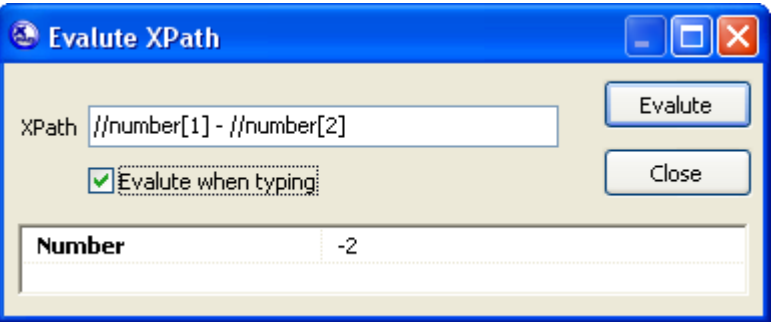
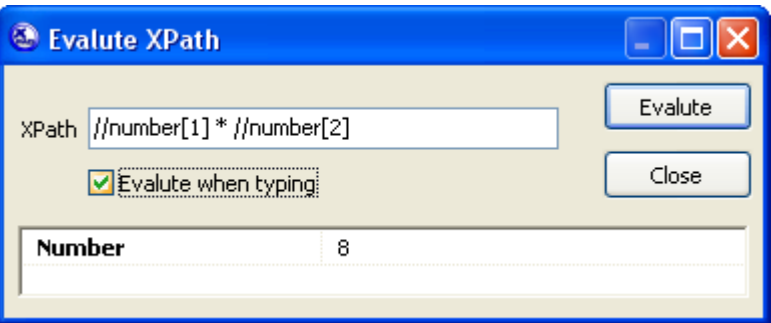
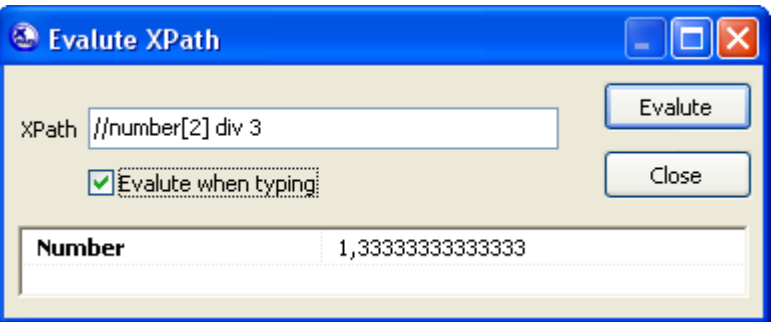
Более подробное описание функций можно найти в спецификации XPath.

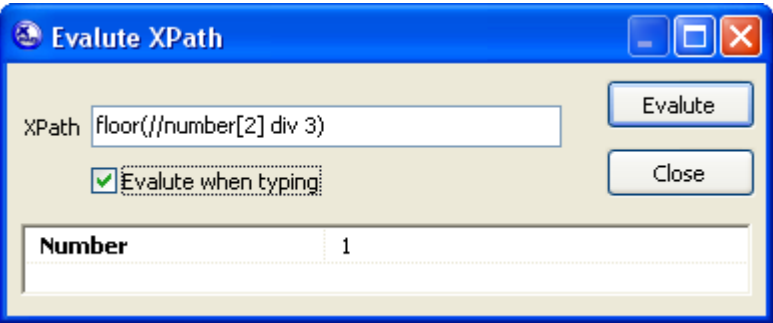
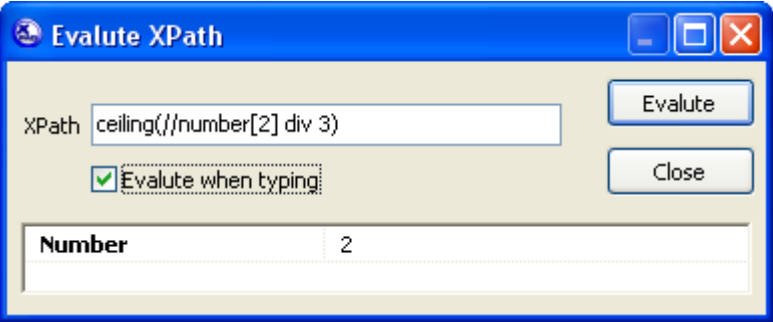
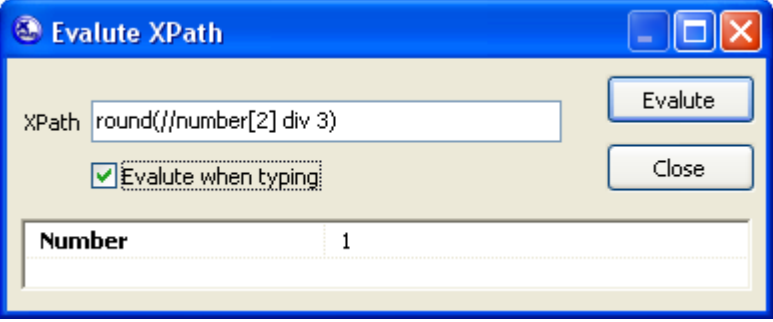
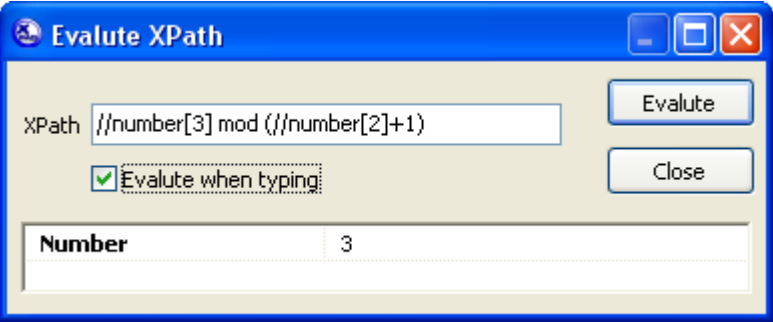
### 2.1.5 Арифметические действия

В XPath предусмотрены основные арифметические действия, а также функции округления. Наверное, выполнение арифметических действий не главная особенность языка запросов, но в некоторых запросах это может быть полезно.



Таблица 2.6. Арифметические действия в XPath.

$//number[1] +$ $//number[2]$	<p>Сложение первого и второго элементов number.</p> 
$//number[1] -$ $//number[2]$	<p>Вычитание.</p> 
$//number[1] *$ $//number[2]$	<p>Умножение.</p> 
$//number[2] \text{ div } 3$	<p>div – деление.</p> 

<p><code>floor ( //number[2] div 3)</code></p>	<p>Функция floor – округление в меньшую сторону до ближайшего целого. Функция может быть использована для получения целой части при делении.</p> 
<p><code>ceiling(//number[2] div 3)</code></p>	<p>Функция ceiling – округление в большую сторону до ближайшего целого.</p> 
<p><code>round(//number[2] div 3)</code></p>	<p>Округление до ближайшего целого.</p> 
<p><code>//number[3] mod (//number[2]+1)</code></p>	<p>mod – получение остатка от деления.</p> 

### 2.1.6 Стандарты XPath 2.0 и XPath 3.0

В предыдущем разделе рассмотрены основы XPath 1.0. Более подробно XPath 1.0 рассмотрен в спецификации [XPath, 1999], а также в [Валиков, 2002].

В январе 2007 года появился стандарт XPath 2.0 [XPath, 2007], который фактически является стандартом XQuery 1.0 [XQuery, 2007] с некоторыми ограничениями. XQuery подробно рассматривается в следующих разделах пособия, при рассмотрении XQuery показаны наиболее существенные отличия от XPath 2.0.

В настоящее время разрабатывается стандарт XPath 3.0 [XPath, 2014], который фактически является стандартом XQuery 3.0 [XQuery, 2014] с некоторыми ограничениями.

Однако стандарт XPath 1.0 не стоит считать устаревшим, так как XPath 2.0 и XPath 3.0 являются расширением XPath 1.0. В большом количестве реализаций технологии XSLT пока используется версия XPath 1.0. XPath 1.0 также используется в технологии XForms.

## 2.2 Технология XSLT 1.0

### 2.2.1 Отображение XML-документа с использованием CSS

Для просмотра документов XML можно использовать таблицы стилей CSS, однако эта технология предоставляет только самые простые возможности форматирования. С ее помощью можно изменить внешний вид, цвет, шрифт, другие параметры элементов, но нельзя, например, изменить порядок вывода элементов.

#### Пример 2.2. Использование CSS для просмотра документа XML.

##### Файл XML:

```
<?xml version="1.0" encoding="Windows-1251"?>
<?xml-stylesheet type="text/css" href="style.css"?>
<!-- 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>
<empty attr1="1" attr2="текст"/>
<CDATA_Example>
    <![CDATA[<<<<<<<<<    >>>>>>>>]]>
</CDATA_Example>
</languages>

```

### Файл CSS:

```

language
{
    background-color : Gray;
    display : block;
}
name
{
    color : Navy;
    font-size : larger;
    font-style : italic;
    font-weight : bold;
}

```

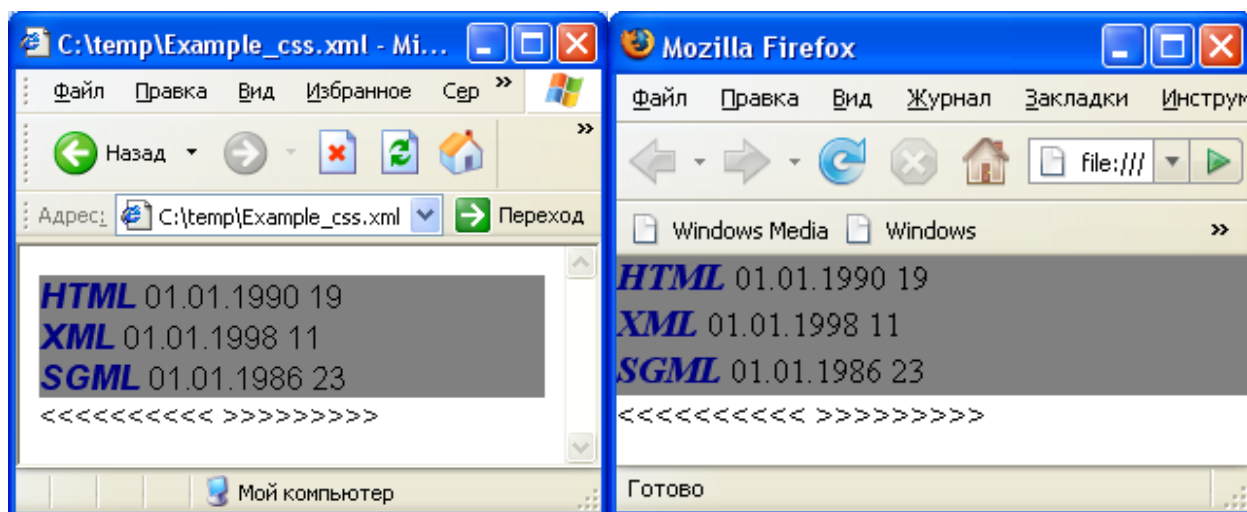


Рис. 2.3. Результат выполнения примера 2.2.

В файле XML наиболее интересна вторая строка:

```
<?xml-stylesheet type="text/css" href="style.css"?>
```

Это инструкция обработки `xml-stylesheet`, в которой задаются параметры отображения документа. Для отображения используется таблица стилей CSS (`type="text/css"`) из файла `"style.css"`. В общем случае в атрибуте `href` задается полный URI (URL) для доступа к таблице стилей.

В файле CSS заданы параметры отображения для элемента `language` (фоновый цвет и отображение с новой строки) и для элемента `name` (цвет текста и параметры шрифта).

Если открыть документ с помощью анализатора XML, который поддерживает инструкцию `xml-stylesheet`, то таблица стилей CSS будет применена к документу. Такие анализаторы, например, встроены в браузеры Mozilla и Internet Explorer.

К сожалению, CSS не позволяет задавать более сложные возможности форматирования. Например, с его помощью нельзя изменить порядок вывода элементов или изменить данные при выводе.

## 2.2.2 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.

С помощью этой технологии можно выполнять преобразования намного более сложные, чем с помощью CSS.

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 запятые). Например, также возможно формирование SQL-сценария на основе XML-данных.

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-процессором. Процессор XSLT 1.0 встроен в большинство современных веб-браузеров.

XSLT-процессор принимает на вход XML-документ, XSLT-преобразование и формирует выходной документ в требуемом формате, как показано на следующем рисунке.

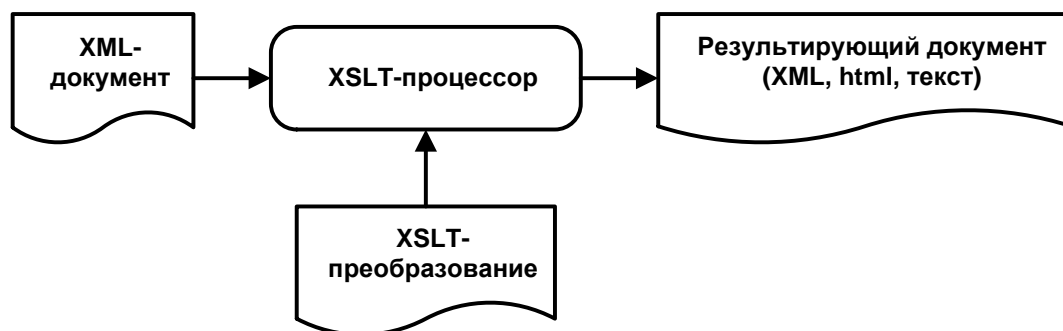


Рис. 2.4. Принцип работы XSLT-процессора.

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.

### 2.2.3 Пример преобразования XSLT

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

#### Пример 2.3.

##### 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>

```

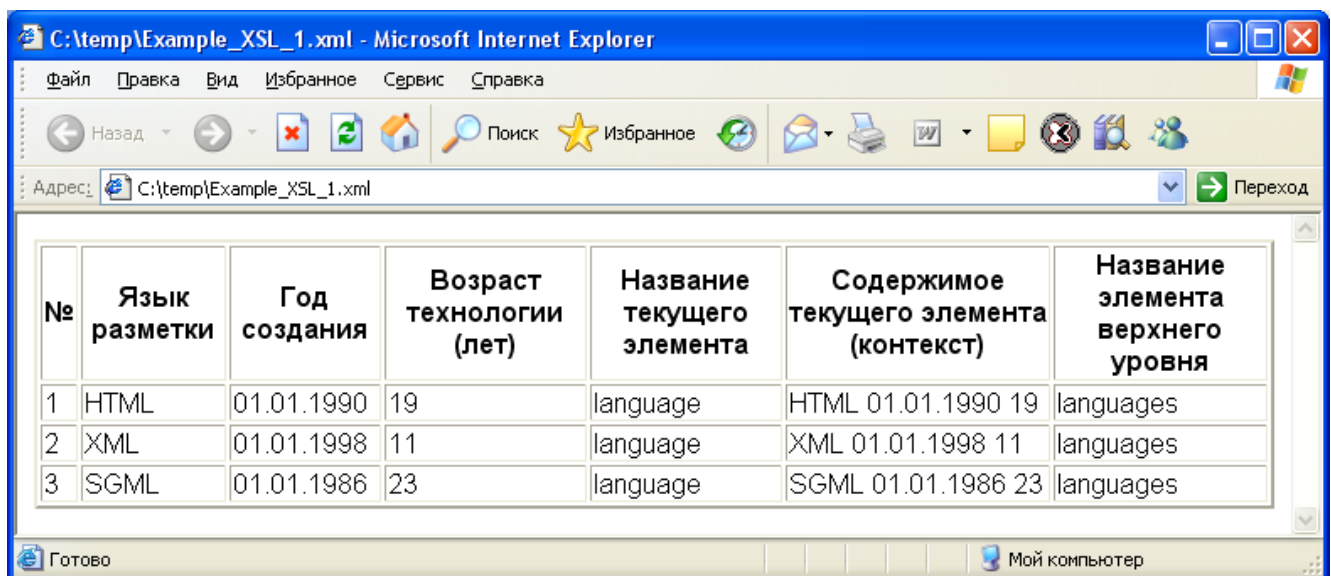


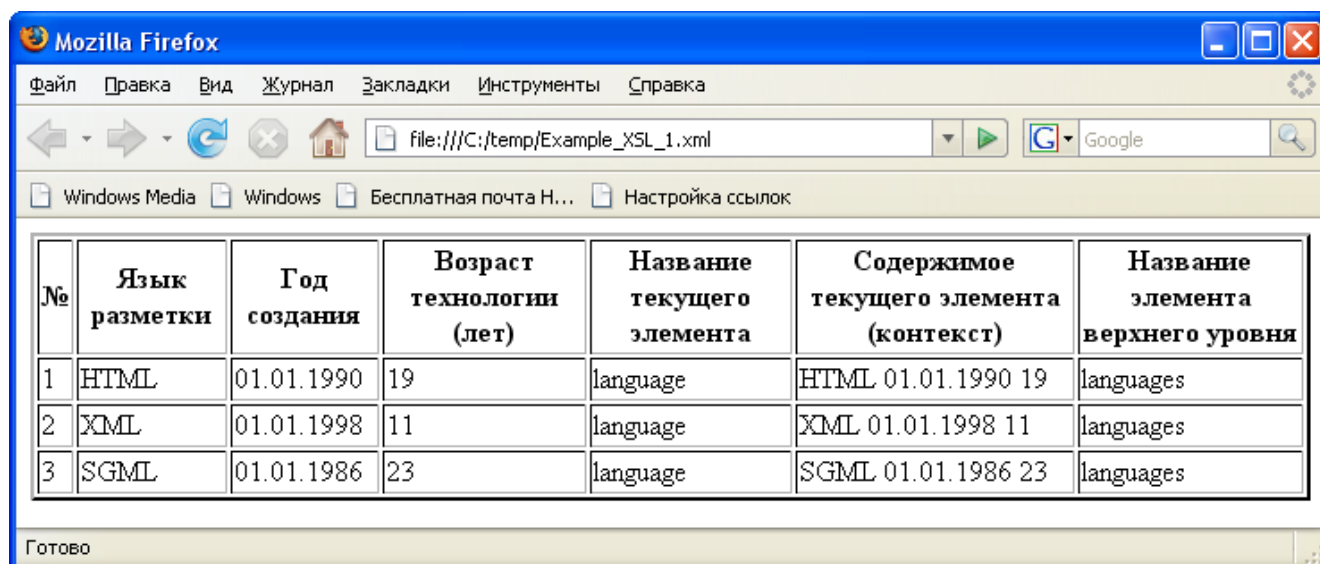
```

<!-- 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:





№	Язык разметки	Год создания	Возраст технологии (лет)	Название текущего элемента	Содержимое текущего элемента (контекст)	Название элемента верхнего уровня
1	HTML	01.01.1990	19	language	HTML 01.01.1990 19	languages
2	XML	01.01.1998	11	language	XML 01.01.1998 11	languages
3	SGML	01.01.1986	23	language	SGML 01.01.1986 23	languages

Рис. 2.5. Результат выполнения примера 2.3.

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

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

Сравним ее с той, которая использовалась для CSS:

```
<?xml-stylesheet type="text/css" href="style.css"?>
```

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, об этом более подробно в следующем разделе.

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>
```

[Оглавление](#)

```

<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.

```

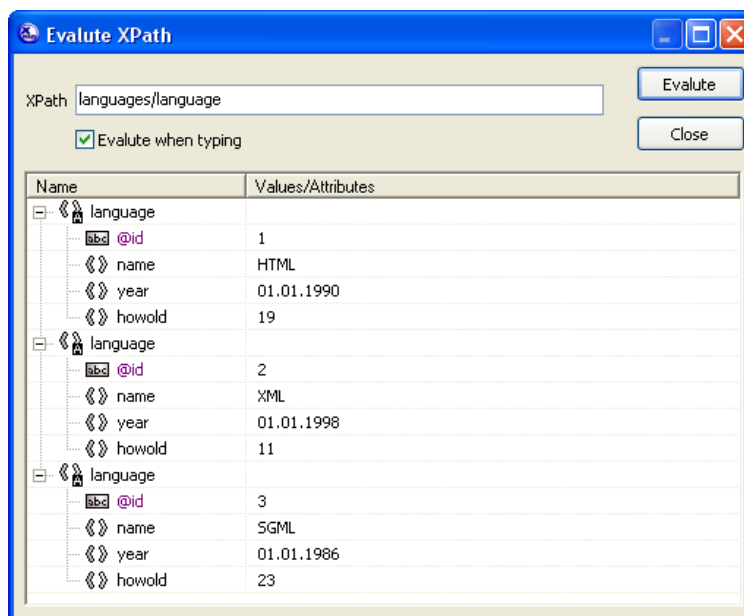
<xsl:for-each select="languages/language">
<!-- Перебор в цикле всех элементов language, вложенных в элемент
languages. -->

```

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.

В нашем примере результат выполнения XPath-выражения languages/language следующий:



The screenshot shows a window titled "Evaluate XPath". The "XPath" field contains the expression "languages/language". The "Evaluate when typing" checkbox is checked. The "Evaluate" button is visible. Below the input field is a table with two columns: "Name" and "Values/Attributes". The table displays the results of the XPath evaluation, showing three "language" elements and their attributes.

Name	Values/Attributes
language	
@id	1
name	HTML
year	01.01.1990
howold	19
language	
@id	2
name	XML
year	01.01.1998
howold	11
language	
@id	3
name	SGML
year	01.01.1986
howold	23

Рис. 2.6. Результат выполнения XPath-выражения «languages/language».

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>

```

Если XML-документ состоит из нескольких секций, то для доступа к ним рекомендуется использовать переменные (`xsl:variable`). Данный пример рассмотрен в разделе «использование переменных».

## 2.2.4 Использование отладчика XSLT в XMLPad

Для использования отладчика необходимо сделать следующее:

1. Открыть преобразование XSLT.
2. В свойстве SourceXML в панели свойств (Properties) выбрать документ XML, содержащий данные для отладки.

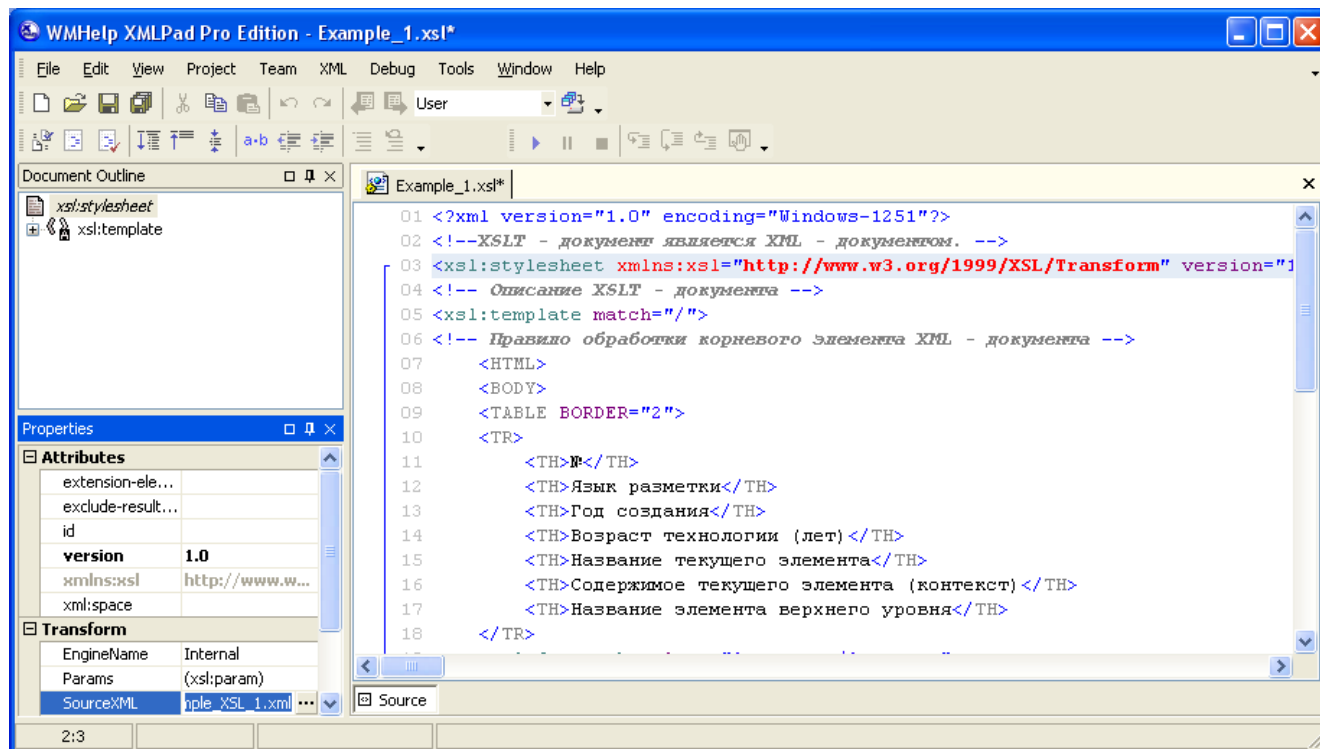


Рис. 2.7. Использование отладчика. Выбор документа XML.

3. Установить контрольные точки, нажимая мышью на область слева от номеров строк.

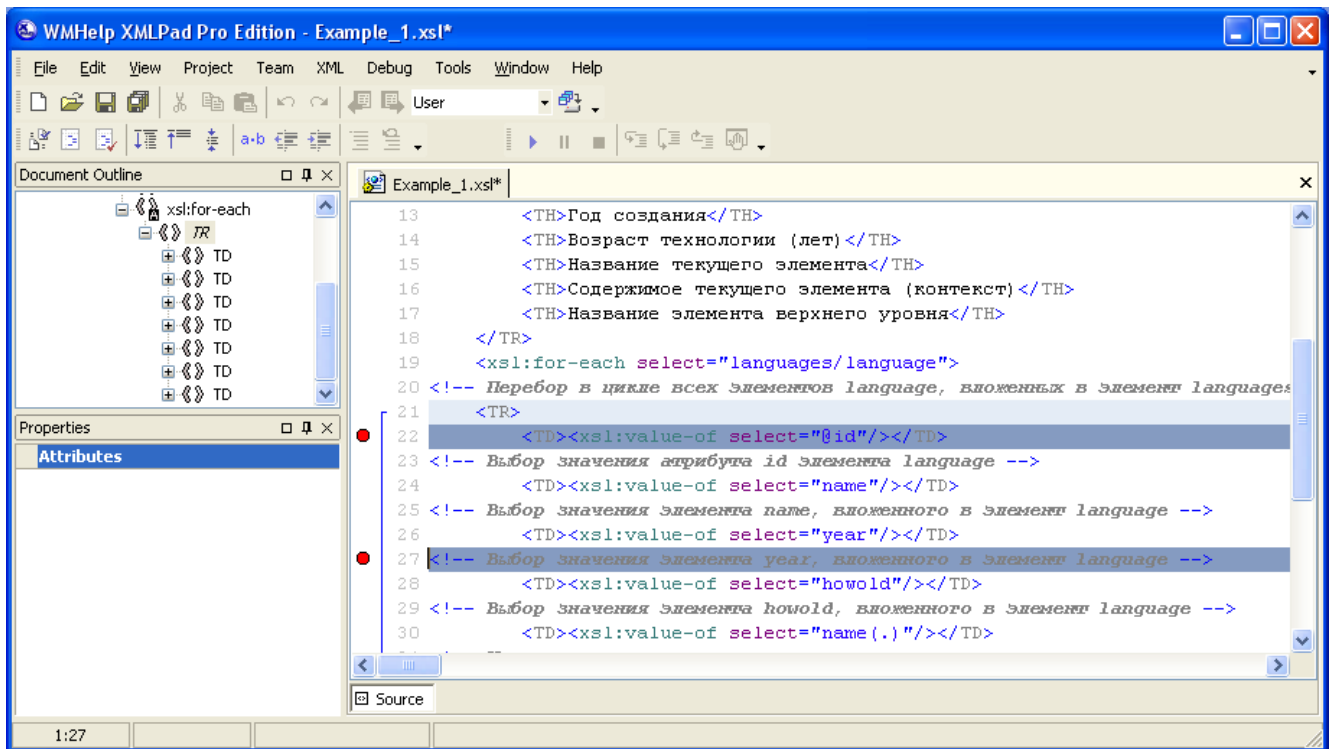


Рис. 2.8. Использование отладчика. Установка контрольных точек.

#### 4. Запустить отладку. (Пункт меню Debug/Run).

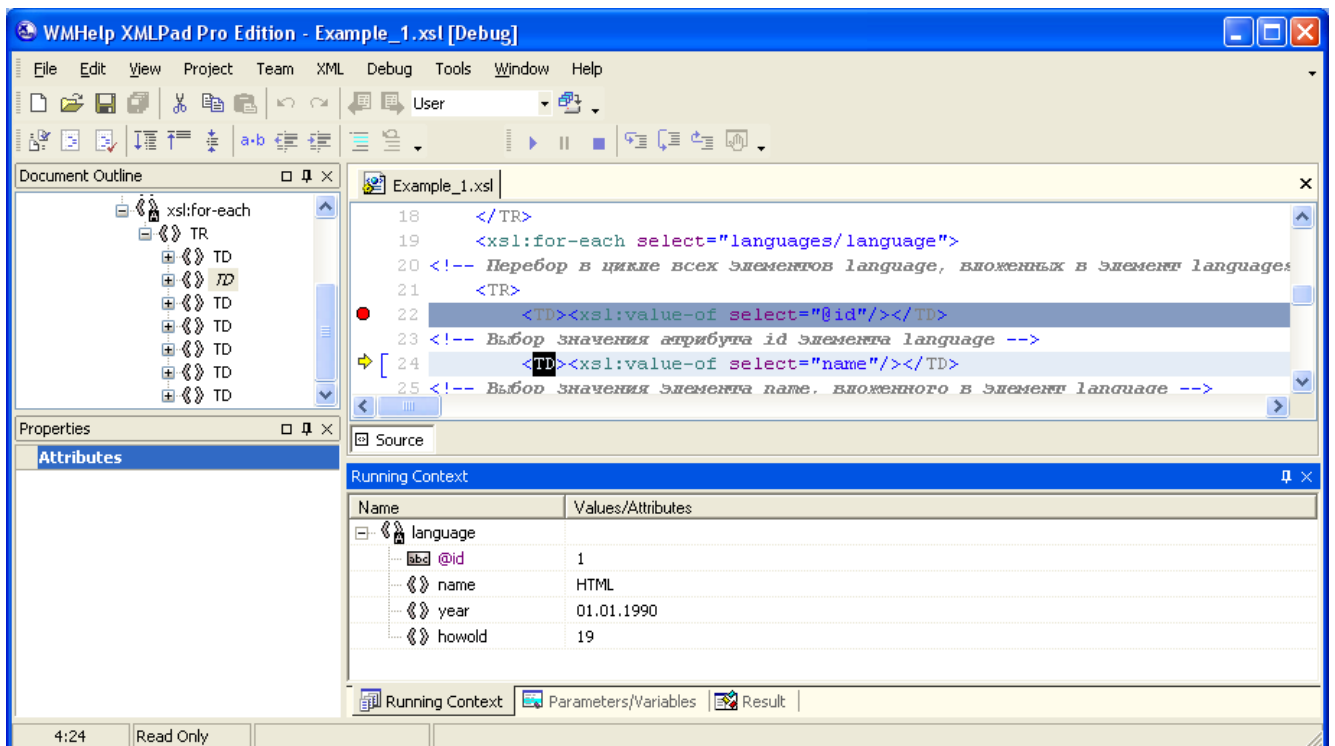


Рис. 2.9. Использование отладчика. Запуск отладки.



В окне Running Context во вкладке Running Context отображается текущий контекст исходного документа. Во вкладке Parameters/Variables отображаются текущие значения параметров и переменных преобразования XSLT. Во вкладке Result отображается документ, который создается в результате преобразования.

5. Для перехода к следующей команде, остановке и перезапуске отладки используются пункты меню Debug.

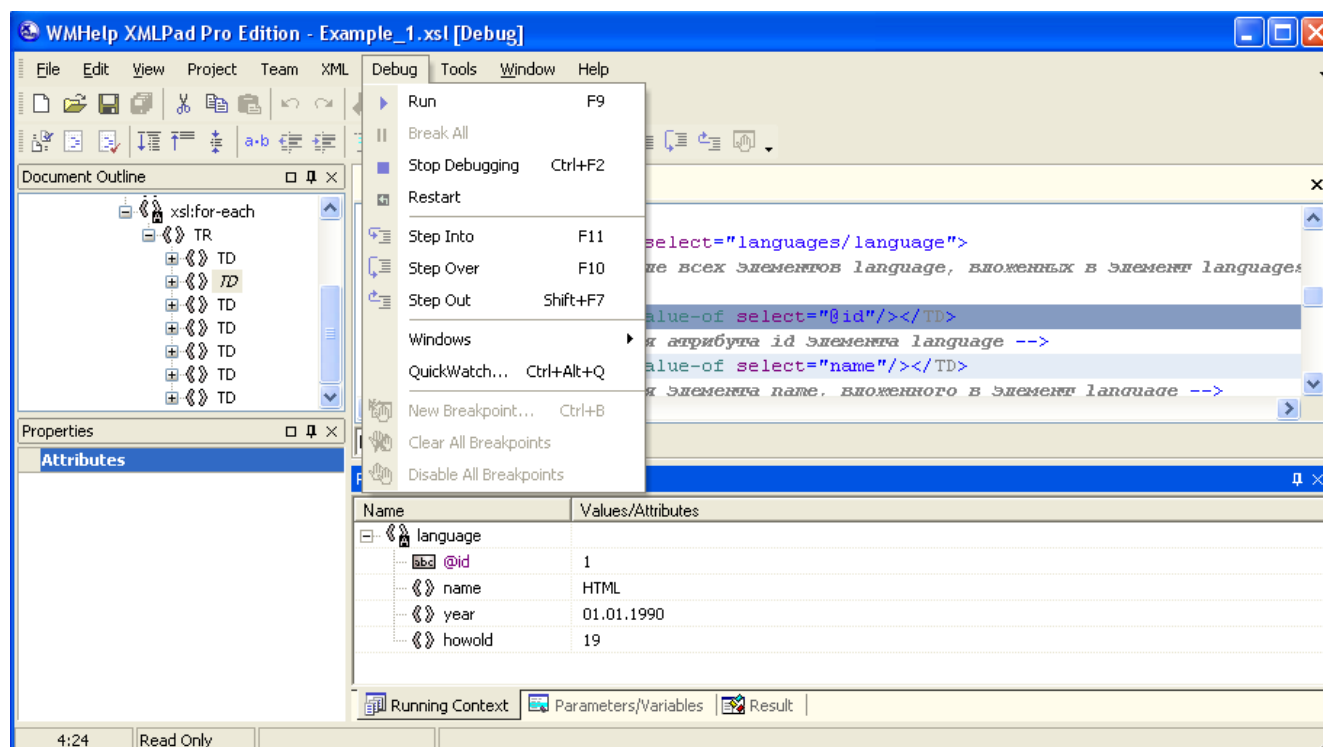


Рис. 2.10. Использование отладчика. Пункты меню Debug.

## 2.2.5 Пространства имен

Атрибут `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"` элемента `stylesheet` определяет пространство имен с префиксом `xsl`. Все элементы, которые начинаются с префикса « `xsl:` », принадлежат данному пространству имен, в данном случае являются командами языка XSLT.

Обратите внимание, что сам элемент

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

уже использует префикс `xsl`, хотя префикс объявляется атрибутом этого элемента. То есть, когда встречается конструкция `xsl:stylesheet`, то префикс «`xsl`:» формально еще не объявлен. Но такая конструкция является допустимой. Префикс пространства можно применять к элементу, в котором он объявляется.

А что обозначает "`http://www.w3.org/1999/XSL/Transform`"? Это значение пространства имен, которое является идентификатором пространства имен. Все XSLT-процессоры должны знать и поддерживать это значение.

Это значение лишь внешне похоже на гиперссылку, такого адреса в Интернет может не существовать. Но все равно это значение будет идентификатором пространства имен.

Если изменить это значение, например, добавить лишние символы после слова `Transform`, то XSLT-процессор выдаст сообщение об ошибке, хотя во всем остальном XSLT-преобразование составлено правильно.

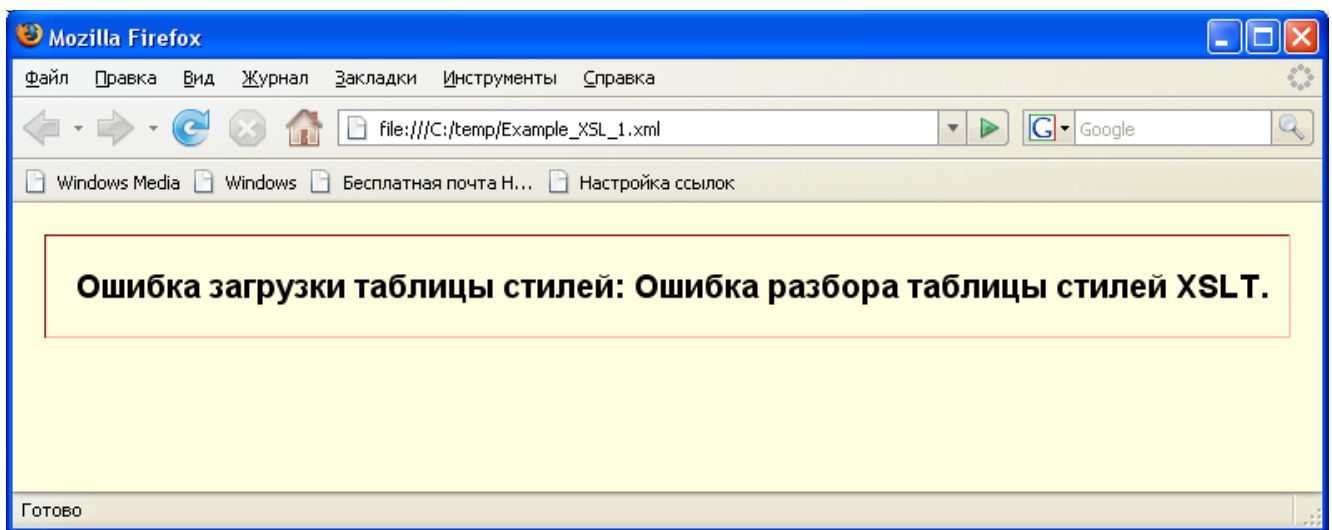


Рис. 2.11. Сообщение об ошибке XSLT-процессора.

Значение пространства имен является идентификатором пространства имен, и его изменять нельзя.

Как правило, в XSLT-преобразованиях используется префикс пространства имен «`xsl`:». Но это не обязательно, и значение префикса «`xsl`:» может быть изменено на любое другое. Заменим префикс пространства имен на «`query`:».

**Пример 2.4. Файл XSLT с префиксом пространства имен `query`:**

```

<?xml version="1.0" encoding="Windows-1251"?>
<!--XSLT - документ является XML - документом. -->
<query:stylesheet xmlns:query="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<!-- Описание XSLT - документа -->
<query: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>
    <query:for-each select="languages/language">
<!-- Перебор в цикле всех элементов language, вложенных в элемент
languages. -->
        <TR>
            <TD><query:value-of select="@id"/></TD>
<!-- Выбор значения атрибута id элемента language -->
            <TD><query:value-of select="name"/></TD>
<!-- Выбор значения элемента name, вложенного в элемент language -->
            <TD><query:value-of select="year"/></TD>
<!-- Выбор значения элемента year, вложенного в элемент language -->
            <TD><query:value-of select="howold"/></TD>
<!-- Выбор значения элемента howold, вложенного в элемент language -
->
            <TD><query:value-of select="name(.)"/></TD>
<!-- Название текущего элемента -->
            <TD><query:value-of select="."/></TD>

```

```

<!-- Содержимое текущего элемента (контекст) -->
    <TD><query:value-of select="name(parent::*)" /></TD>
<!-- Название элемента верхнего уровня (также можно использовать
select="name(..)") -->
    </TR>
</query:for-each>
</TABLE>
</BODY>
</HTML>
</query:template>
</query:stylesheet>

```

### Результат работы в браузере:

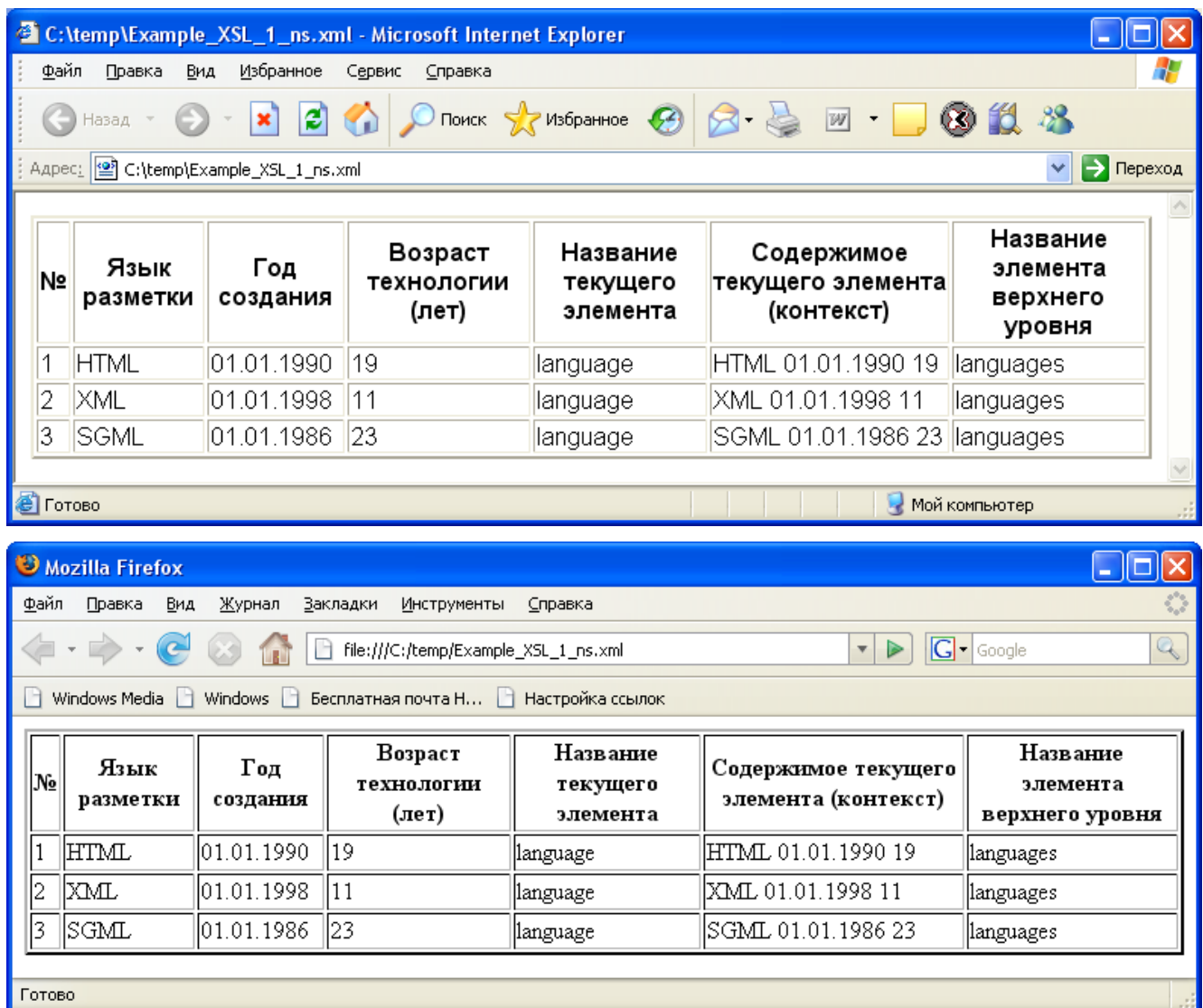


Рис. 2.12. Результат выполнения примера 2.4.

Несмотря на изменение префикса пространства имен, преобразование работает. Но если изменить значение пространства имен, то преобразование не будет работать.

### 2.2.6 Шаблоны с несколькими элементами `template`

Рассмотренный шаблон XSLT является шаблоном с жестко заданной 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 `templat`'ов не задается. Они вызываются в такой последовательности, в которой встречаются соответствующие элементы во входном документе.

#### Пример 2.5.

##### 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:

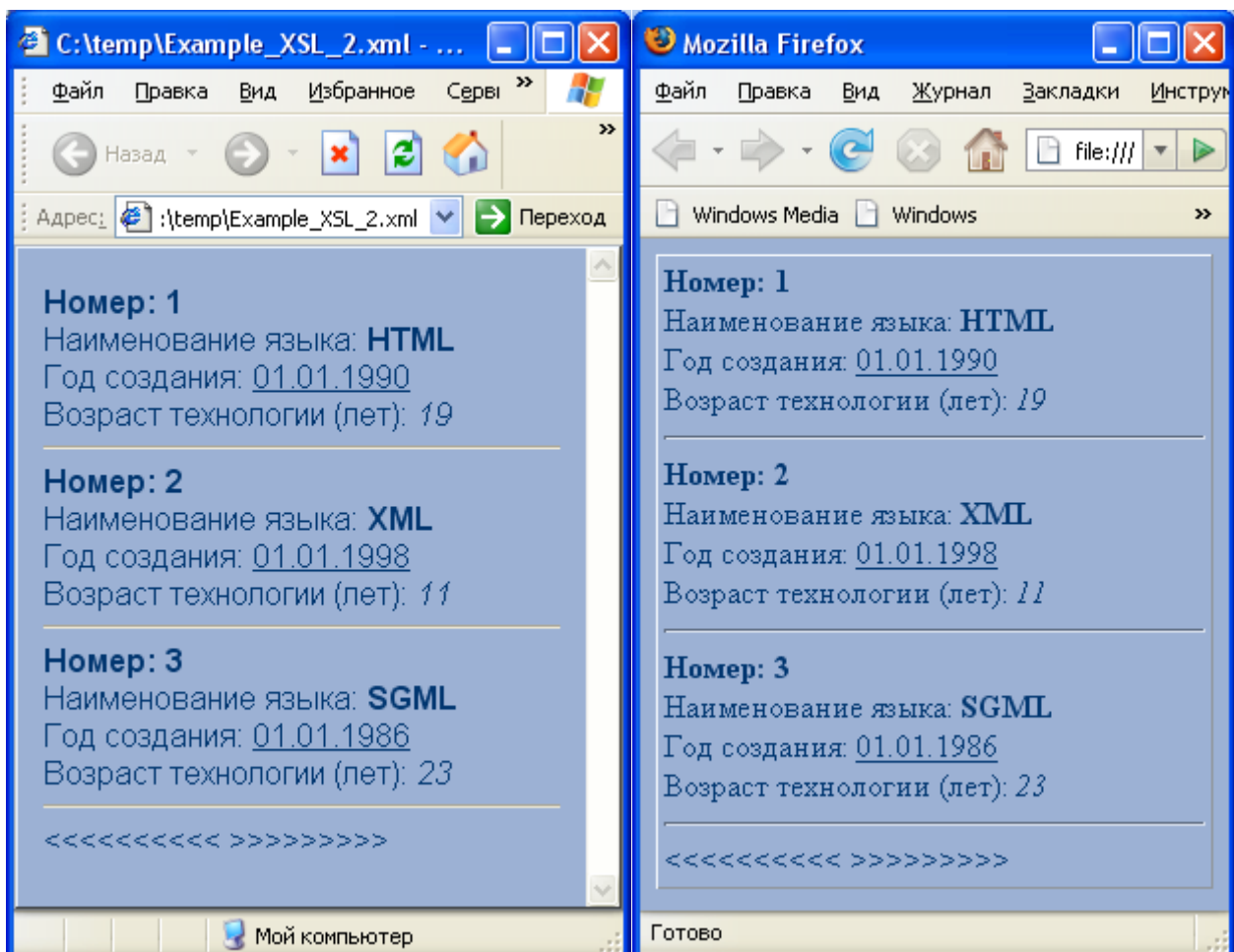


Рис. 2.13. Результат выполнения примера 2.5.

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 пытается 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. Инstrukция 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».

Таким образом, 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.

Например, если в предыдущем примере указать

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

то будут вызваны шаблоны только для элементов name:

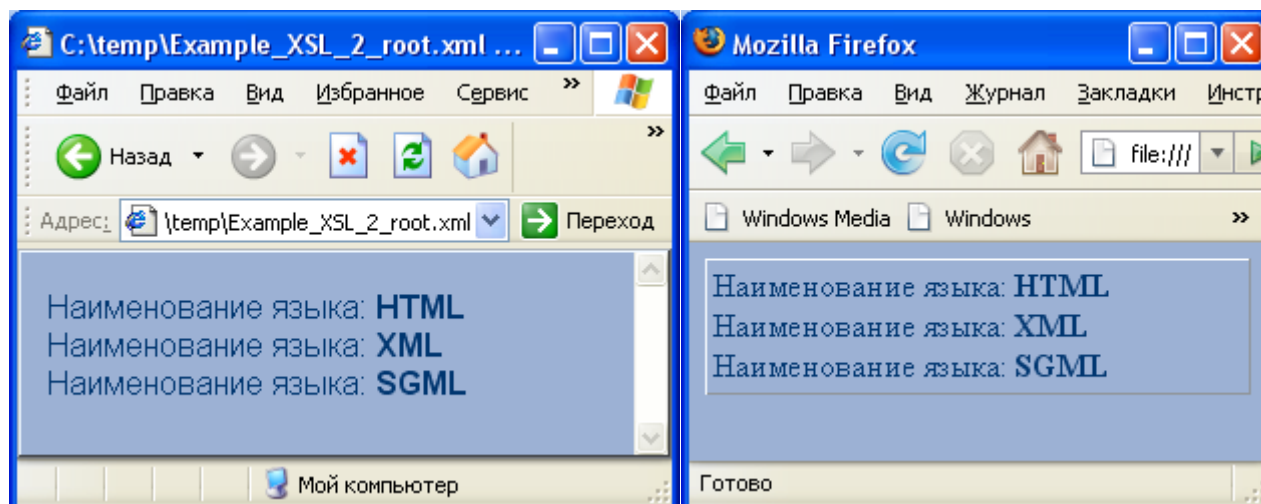


Рис. 2.14. Результат выполнения примера 2.5. Изменение 1.

Еще одна интересная особенность команды apply-templates состоит в том, что она умеет «пропускать» пустые элементы в иерархии. Например, если из преобразования удалить шаблоны для languages и language, то команда apply-templates в шаблоне для корневого элемента будет сразу находить шаблоны для элементов name, year и howold. Просмотр результата в браузере:



Если в исходном документе поменять местами элементы `name` и `year`, то это приведет к перестановке соответствующих строк в выходном HTML-документе, так как шаблоны для этих элементов будут вызываться в другой последовательности. Просмотр результата в браузере:

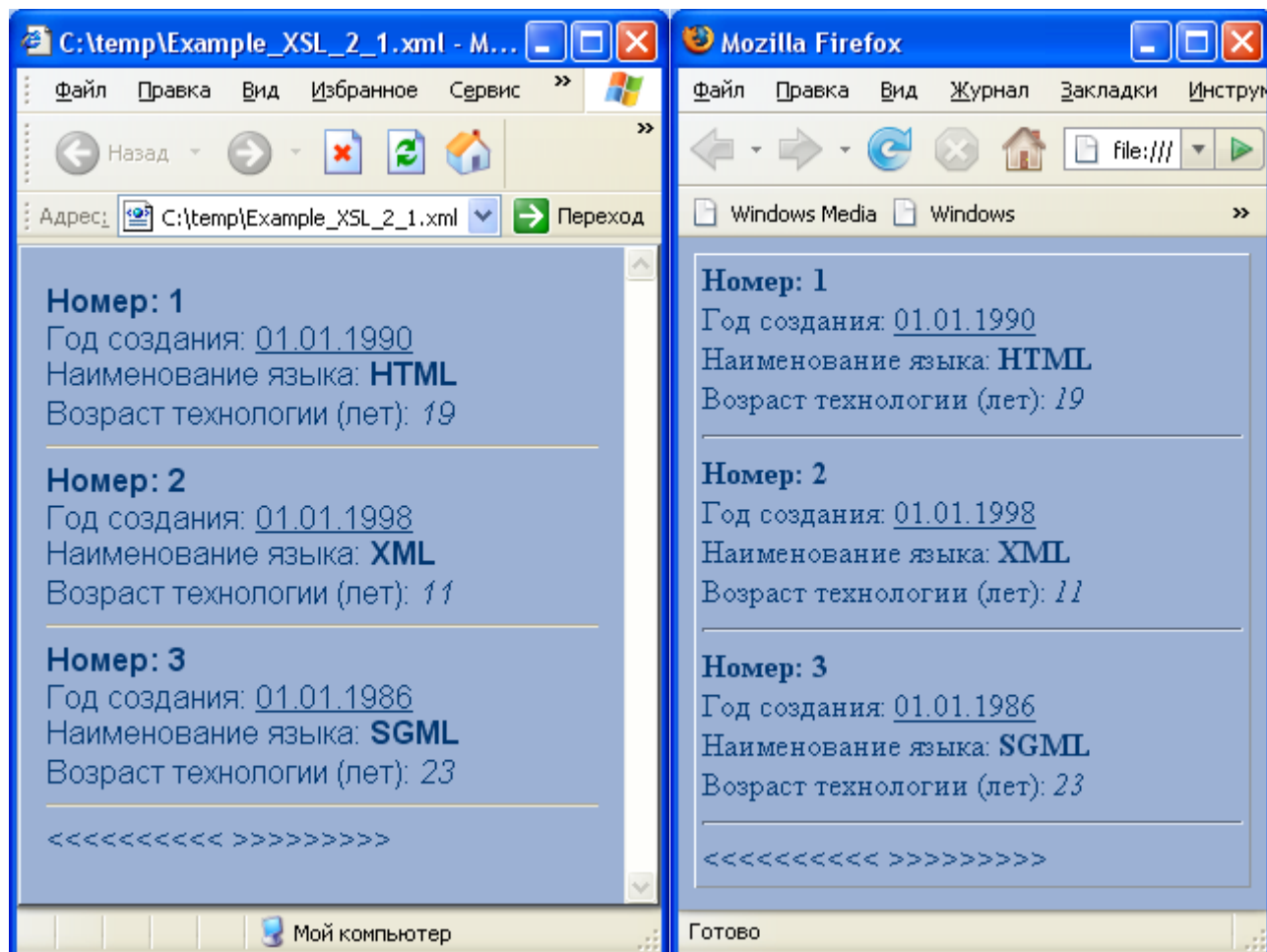


Рис. 2.17. Результат выполнения примера 2.5. Изменение 4.

Следующее XSLT-преобразование является модификацией предыдущего, в нем используются условные операторы, операторы для создания элементов и атрибутов.

### Пример 2.6.

#### 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:comment>Комментарий в выходном документе</xsl:comment>
    <xsl:apply-templates select="//language"/>

    <!-- Создание элемента с помощью element -->
    <xsl:element name="P" use-attribute-sets="p_attrs">
        <xsl:attribute name="align">center</xsl:attribute>
        Параграф текста
    </xsl:element>

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

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

<!-- Получение значения атрибута id (префикс @ означает атрибут) -->
    <xsl:choose>
        <xsl:when test="name[.='XML']">
            <B><I>Номер: <xsl:value-of select="@id"/></I></B><BR/>
        </xsl:when>
        <xsl:otherwise>
            Номер: <xsl:value-of select="@id"/><BR/>
        </xsl:otherwise>
    </xsl:choose>

```

```

</xsl:choose>

<!-- Вызов шаблонов для элементов name, year -->
<xsl:apply-templates select="./name"/>
<xsl:apply-templates select="year"/>

<!-- Вызов шаблона с помощью xsl:call-template для элемента
howold -->
<xsl:call-template name="Howold_Function">
  <!--<xsl:with-param name="ParamHowold" select="./howold"/>-
->
  <xsl:with-param name="ParamHowold">
    <xsl:value-of select="howold"/>
  </xsl:with-param>
</xsl:call-template>
<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>
<!-- ++++++ -->

<!-- ++++++ -->
<!-- xsl:if - проверка простых условий без ELSE -->
<!-- Шаблон обработки элемента year -->
<xsl:template match="year">
  <div>
    <xsl:if test="number(substring(.,7,4)) < number('1995')">
      <xsl:attribute name="align">right</xsl:attribute>
    </xsl:if>

```



```

        Год создания: <U><xsl:value-of select="."/></U><BR/>
    </div>
</xsl:template>
<!-- ++++++ -->

<!-- ++++++ -->
<!-- Шаблон обработки элемента howold -->
<xsl:template name="Howold_Function">
    <xsl:param name="ParamHowold" select="0"/>

    <I>Возраст технологии (лет): <xsl:value-of
select="$ParamHowold"/></I><BR/>
</xsl:template>
<!-- ++++++ -->

<xsl:attribute-set name="p_attrs">
    <xsl:attribute name="title">Подсказка</xsl:attribute>
    <xsl:attribute name="onclick">alert('Подсказка')</xsl:attribute>
</xsl:attribute-set>
<!-- ++++++ -->

</xsl:stylesheet>

```

**Просмотр результата в браузере:**

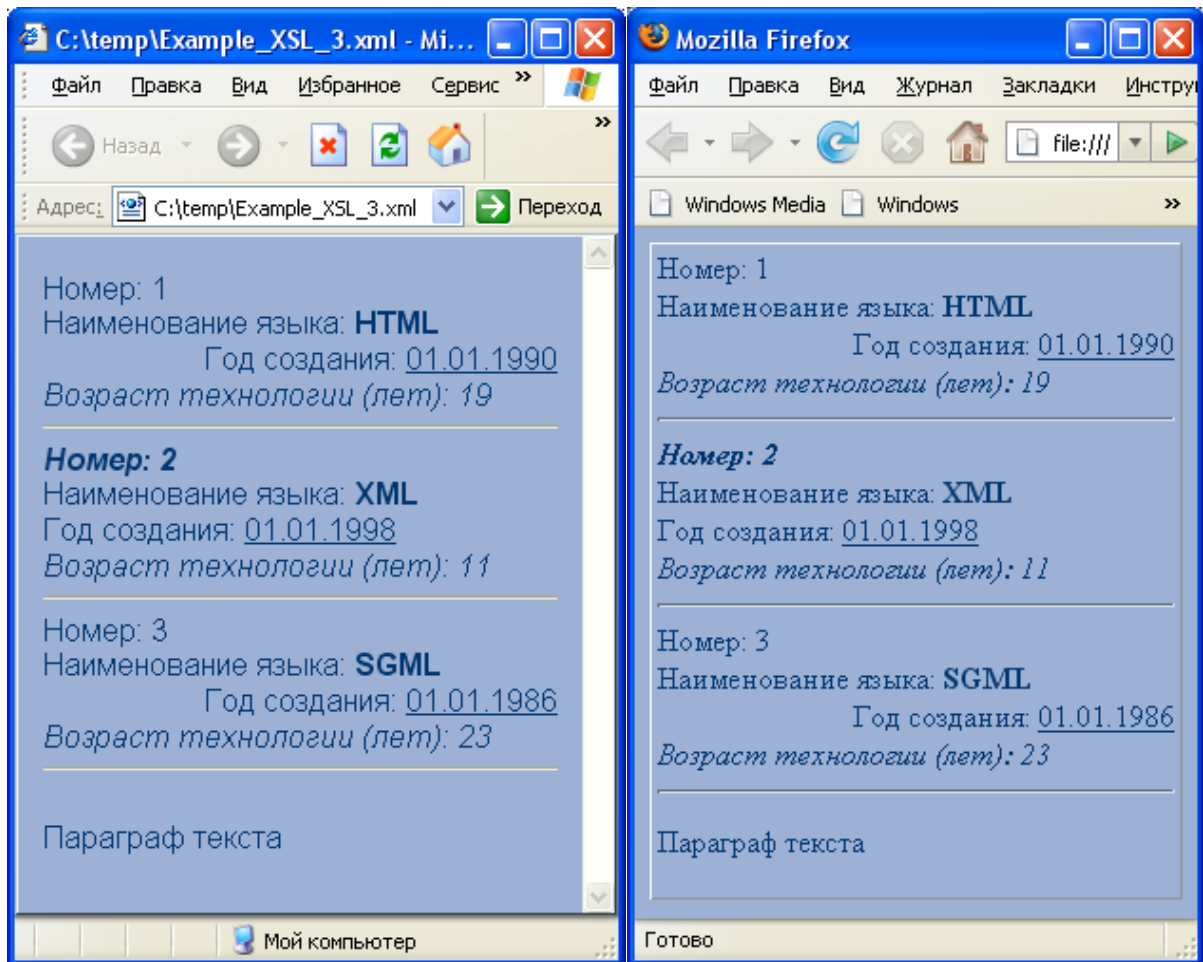


Рис. 2.18. Результат выполнения примера 2.6.

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="/">
```

```
  <HTML>
```

```
  <HEAD>
```

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

```
  </HEAD>
```

```
<BODY>
```

```
<xsl:comment>Комментарий в выходном документе</xsl:comment>
```

Создание комментария в выходном HTML-документе. В документе появится узел `<!-- Комментарий в выходном документе -->`

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

Шаблоны вызываются только для элементов `language`. Поэтому, в отличие от предыдущего примера, содержимое других элементов не выводится в выходной поток.

```
<!-- Создание элемента с помощью element -->
<xsl:element name="P" use-attribute-sets="p_attrs">
  <xsl:attribute name="align">center</xsl:attribute>
  Параграф текста
</xsl:element>
```

Команда `element` создает элемент в выходном документе. В данном примере это элемент `P`. Для него создается атрибут `align="center"`, а также атрибуты из набора `p_attrs`.

Обычно с помощью команды `element` создают сложные вычисляемые элементы, которые не могут быть созданы простым указанием тэга.

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

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

<!-- Получение значения атрибута id (префикс @ означает атрибут) -->
<xsl:choose>
  <xsl:when test="name[.='XML']">
    <B><I>Номер: <xsl:value-of select="@id"/></I></B><BR/>
```

```

</xsl:when>
<xsl:otherwise>
    Номер: <xsl:value-of select="@id"/><BR/>
</xsl:otherwise>
</xsl:choose>

```

Условный оператор `choose` может содержать множество вложенных элементов проверки условия `when`, а также элемент `otherwise`, который срабатывает, если не выполняется ни одно условие.

Условие указывается в атрибуте `test` оператора `when`. В примере проверяется, что текущее значение элемента `name='XML'` (`name[.='XML']`).

Условие представляет собой XPath-выражение, которое должно возвращать логическое значение. Если условие истинно, то выполняются операторы, вложенные в `when`.

```

<!-- Вызов шаблонов для элементов name, year -->
<xsl:apply-templates select="./name"/>
<xsl:apply-templates select="year"/>

<!-- Вызов шаблона с помощью xsl:call-template для элемента
howold -->
<xsl:call-template name="Howold_Function">
<!--<xsl:with-param name="ParamHowold" select="./howold"/>-->
    <xsl:with-param name="ParamHowold">
        <xsl:value-of select="howold"/>
    </xsl:with-param>
</xsl:call-template>

```

Шаблон `Howold_Function` вызывается по имени, как функция. С помощью команды `with-param` передаются параметры вызова. В атрибуте `name` элемента `with-param` указывается название параметра. Значение параметра может быть задано содержимым элемента или указано в атрибуте `select`.

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

<!-- ++++++ -->
<!-- xsl:if - проверка простых условий без ELSE -->
<!-- Шаблон обработки элемента year -->
<xsl:template match="year">
    <div>
        <xsl:if test="number(substring(.,7,4)) < number('1995')">
            <xsl:attribute name="align">right</xsl:attribute>
        </xsl:if>

```

Условный оператор if проверяет условие, указанное в атрибуте test. Если условие истинно, то выполняются операторы, вложенные в if. Оператор if в XSLT не содержит конструкции ELSE. Поэтому вместо него, как правило, используют оператор choose.

В нашем примере проверяется, что подстрока, полученная из строки года меньше чем 1995. Вместо оператора сравнения «<» пишется ссылка на символ &lt;

```

    Год создания: <U><xsl:value-of select="."/></U><BR/>
</div>
</xsl:template>
<!-- ++++++ -->

<!-- ++++++ -->
<!-- Шаблон обработки элемента howold -->

```

```
<xsl:template name="Howold_Function">
  <xsl:param name="ParamHowold" select="0"/>
```

В этом примере у шаблона используется не атрибут `match`, а атрибут `name`. Такой шаблон вызывается как функция. Элемент `param` объявляет параметры, которые могут передаваться в шаблон.

У элемента `xsl:param` атрибут `name` задает название параметра, а атрибут `select` значение по-умолчанию.

```
<I>Возраст технологии (лет): <xsl:value-of
select="$ParamHowold"/></I><BR/>
```

Знак `$` ставится для обозначения обращения к параметру или переменной.

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

<xsl:attribute-set name="p_attrs">
  <xsl:attribute name="title">Подсказка</xsl:attribute>
  <xsl:attribute name="onclick">alert('Подсказка')</xsl:attribute>
</xsl:attribute-set>
```

Набор атрибутов `p_attrs` содержит два атрибута: `title` и `onclick`. Далее все атрибуты из этого набора могут быть вставлены в элемент с помощью команды `<xsl:element name=" название элемента " use-attribute-sets=" название набора атрибутов ">`.

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

### 2.2.7 Включение стилей

В соответствии со спецификацией [XSLT, 1999], «механизм включения стилей позволяет объединять стили без изменения семантики комбинированных стилей».

Для включения стилей используется элемент `xsl:include`.

Рассмотрим пример включения стилей:

### Пример 2.7.

#### Файл XML:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="include.xsl"?>
<ops>
  <op name="add" symbol="+">
    <operand>1</operand>
    <operand>2</operand>
  </op>
</ops>
```

#### Файл include.xsl:

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

  <xsl:include href="math_include.xsl"/>
```

Подключение файла с дополнительными шаблонами.

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

В шаблоне для корневого элемента в результате применения `apply-templates` будет вызван шаблон из файла `math_include.xsl`.

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

#### Файл math\_include.xsl:

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

```

<xsl:template match="op[@symbol='+' ]">
  <xsl:value-of select="operand[1]" />
  <xsl:value-of select="@symbol" />
  <xsl:value-of select="operand[2]" />
=
  <xsl:value-of select="sum(operand)" />
</xsl:template>
</xsl:stylesheet>

```

### Просмотр результата в браузере:

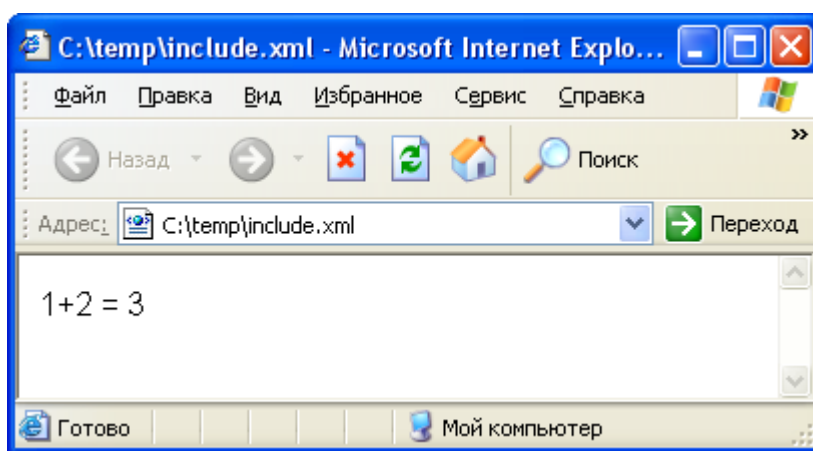


Рис. 2.19. Результат выполнения примера 2.7.

Элемент `include` позволяет включать шаблоны друг в друга. Это может быть полезно в том случае, когда необходимо создать библиотеку шаблонов в отдельном файле и подключать ее в различные XSLT-преобразования.

### 2.2.8 Импорт стилей

В соответствии со спецификацией [XSLT, 1999], «механизм импорта стилей позволяет стилям переписывать друг друга».

Для импорта стилей используется элемент `xsl:import`.

Рассмотрим пример импорта стилей:

#### Пример 2.8.

#### Файл XML:



```

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="import.xsl"?>
<ops>
  <op name="add" symbol="+">
    <operand>1</operand>
    <operand>2</operand>
  </op>
  <op name="sub" symbol="-">
    <operand>1</operand>
    <operand>2</operand>
  </op>
  <op name="mul" symbol="*">
    <operand>1</operand>
    <operand>2</operand>
  </op>
</ops>

```

### Файл import.xsl:

```

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

```

```

  <xsl:import href="math.xsl"/>

```

```

  <!-- С каждой следующей операцией import приоритет повышается -->
  <xsl:import href="string.xsl"/>

```

Так как с каждой следующей операцией import приоритет шаблонов повышается, то шаблоны в файле string.xsl будут иметь больший приоритет, чем шаблоны в файле math.xsl

```

<xsl:template match="op">
  <xsl:value-of select="operand[1]"/>
  <xsl:value-of select="@symbol"/>
  <xsl:value-of select="operand[2]"/>

```

```
= <xsl:apply-imports/><br/>
```

Команда `apply-imports` аналогична команде `apply-templates`, но она вызывает шаблоны, которые импортированы с помощью `import` с учетом приоритета импортируемых файлов.

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

### Файл **math.xml**:

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

  <xsl:template match="op[@symbol='+' ]">
    <xsl:value-of select="sum(operand)"/> (математическое)
  </xsl:template>

  <xsl:template match="op[@symbol='-']">
    <xsl:value-of select="number(operand[1])-number(operand[2])"/>
    (математическое)
  </xsl:template>

  <xsl:template match="op[@symbol='*']">
    <xsl:value-of select="number(operand[1])*number(operand[2])"/>
    (математическое)
  </xsl:template>

</xsl:stylesheet>
```

### Файл **string.xml**:

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

```
<xsl:template match="op[@name='add']">
  <xsl:value-of select="operand[1]"/>
  <xsl:value-of select="operand[2]"/>
  (строковое)
</xsl:template>
```

```
<xsl:template match="op[@name='mul']">
  <xsl:value-of select="operand[2]"/>
  <xsl:value-of select="operand[1]"/>
  (строковое)
</xsl:template>
```

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

### Просмотр результата в браузере:

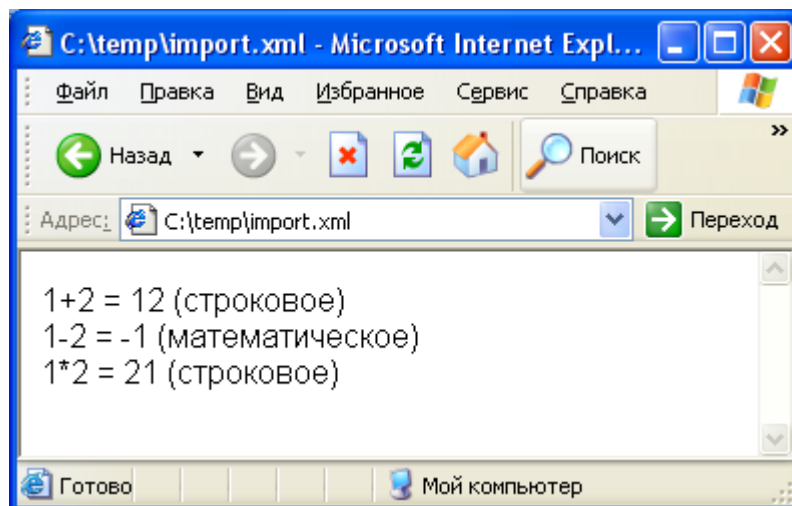


Рис. 2.20. Результат выполнения примера 2.8.

Файл `math.xml` содержит шаблоны для всех трех элементов `op`, а файл `string.xml` содержит шаблоны только для двух элементов. Так как строка

```
<xsl:import href="string.xml"/>
```

указана в файле преобразования позднее, чем строка

```
<xsl:import href="math.xml"/>
```

то «строковые» шаблоны имеют больший приоритет, чем «математические».

Если строки поменять местами, то результат выполнения будет следующим:

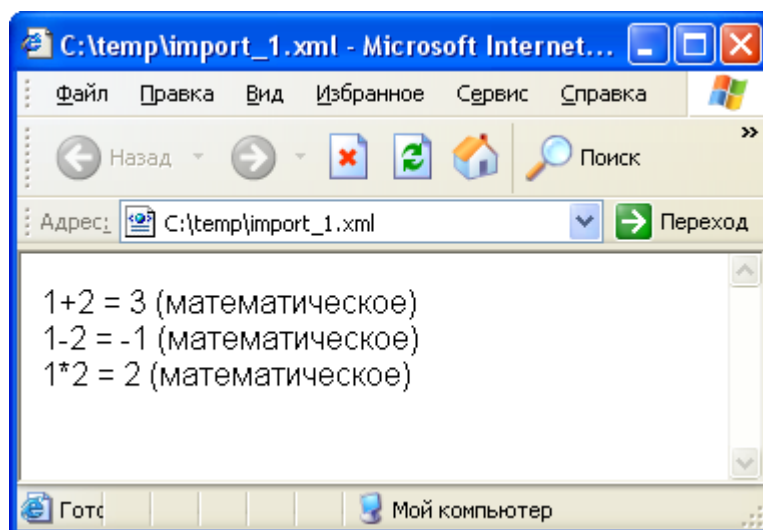


Рис. 2.21. Результат выполнения примера 2.8. Изменение 1.

В этом случае выполняются все «математические» шаблоны, так как они имеют больший приоритет, а «строковые» не выполняются.

### 2.2.9 Использование сортировки

Для сортировки используется элемент `sort`. Этот элемент может быть вложен в элемент `for-each` (определяет порядок перебора элементов в цикле) или в элемент `apply-templates` (определяет порядок вызова шаблонов для элементов).

Для элемента `sort` используются следующие атрибуты:

- `select` = XPath-выражение, определяет выражение, по которому производится сортировка.
- `data-type` = "text" | "number", тип данных текстовый или числовой.
- `order` = "ascending" | "descending", порядок сортировки: по возрастанию или по убыванию.
- `case-order` = "upper-first" | "lower-first", в случае текстовой сортировки определяет порядок сортировки заглавных букв. "upper-first" – заглавные буквы размещаются первыми.

В примере также используется элемент `variable` для создания переменной. Элемент `variable` может быть использован в виде:

```
<xsl:variable name="название" select="значение"/>
```

или

```
<xsl:variable name="название">значение</xsl:variable>
```

Для обращения к переменным перед названием ставится знак \$, например:

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

Основной особенностью переменных в XSLT является то, что их нельзя изменять, фактически они являются константами. Наиболее часто переменные используют для удобства программирования, занося в них промежуточные фрагменты документов.

## Пример 2.9.

### Файл XSLT:

```
<?xml version="1.0" encoding="Windows-1251"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<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>
  </TR>

  <xsl:variable name="LangVar" select="languages"/>
  <!-- В переменную LangVar помещается значение элемента
languages -->

  <xsl:for-each select="$LangVar/language">
    <xsl:sort select="howold" data-type="number"
order="ascending"/>
```

```

<!-- Перебор в цикле всех элементов language, вложенных в элемент
languages. Сортировка по возрастанию значения элемента howold. -->
    <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 -
->
    </TR>
</xsl:for-each>

    <xsl:for-each select="$LangVar/language">
        <xsl:sort select="howold" data-type="number"
order="descending"/>
<!-- Перебор в цикле всех элементов language, вложенных в элемент
languages. Сортировка по убыванию howold. -->
        <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 -
->
        </TR>
    </xsl:for-each>
</TABLE>

<hr/>

```

```

    <xsl:apply-templates select="languages/language">
        <xsl:sort select="name" data-type="text" case-
order="upper-first"/>
        <!-- Сортировка первого уровня по элементу name -->
        <xsl:sort select="howold" data-type="number"
order="descending"/>
        <!-- Если значения элементов name совпадают, то сортировка
второго уровня по элементу howold -->
        <!-- Элементы sort вложены в элемент apply-templates -->
    </xsl:apply-templates>

</BODY>
</HTML>
</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:**



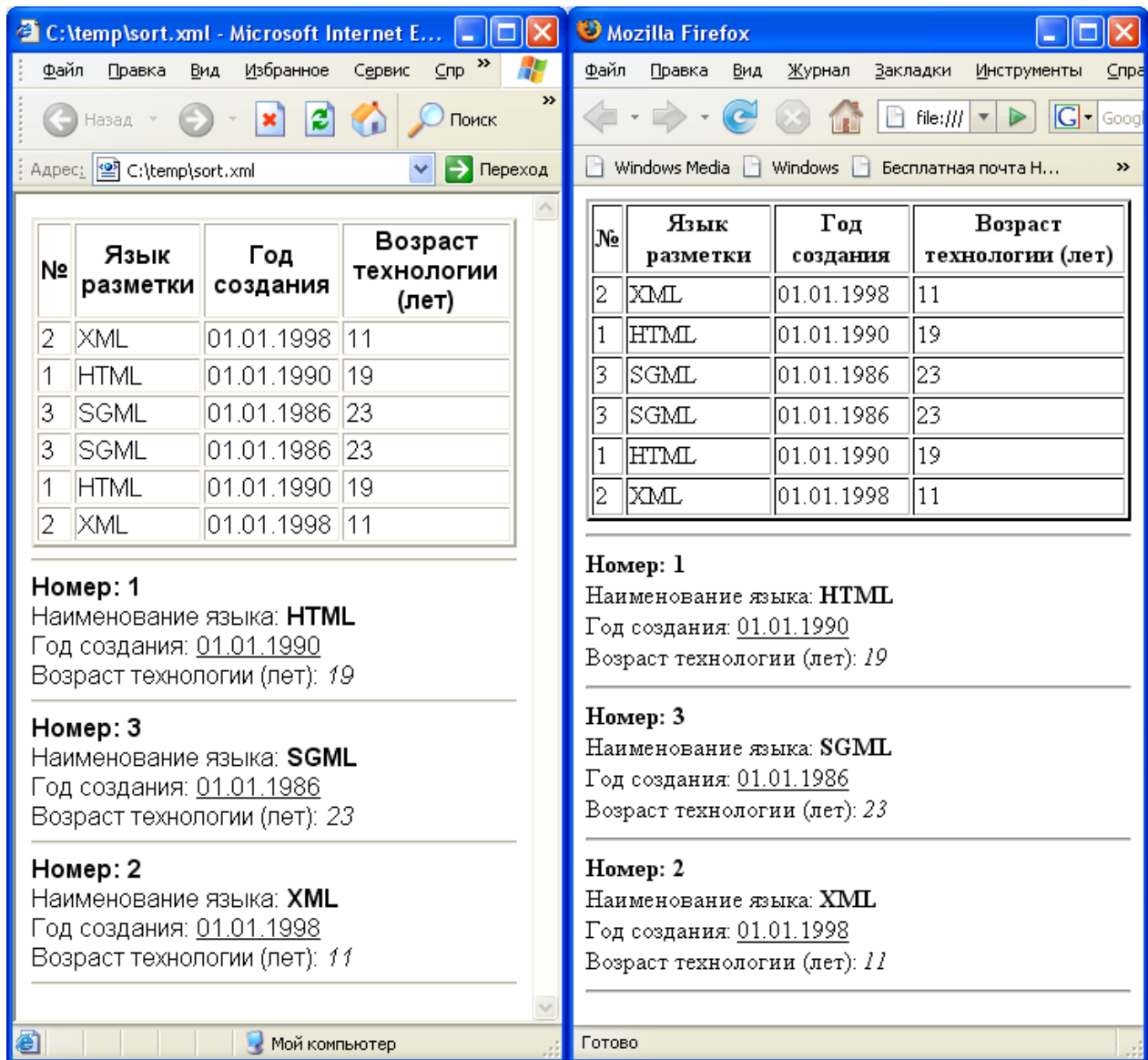


Рис. 2.22. Результат выполнения примера 2.9.

### 2.2.10 Формирование текстового документа

Для указания типа выходного документа используется элемент `xsl:output` с атрибутом `method="тип документа"`.

В следующем примере формируется текстовый документ:

#### Пример 2.10.

##### Файл XSLT:

```
<?xml version="1.0" encoding="Windows-1251"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
```

```
<xsl:output method="text" encoding="Windows-1251"/>
```

```
<!-- output определяет формат выходного документа.
```

Атрибут method="text" - выходной документ задается как текст  
(возможны также значения "xml", "html")

```
Атрибут encoding задает кодировку выходного документа -->
```

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

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

```
    <xsl:for-each select="languages/language">
```

```
        <!-- Перебор в цикле всех элементов language, вложенных в  
элемент languages. -->
```

```
            <xsl:text>&lt;</xsl:text>
```

```
            <!-- text формирует текстовый узел в выходном документе.
```

```
В этом примере формируется тэговая скобка -->
```

```
            <xsl:value-of select="@id"/>,</xsl:for-each>
```

```
            <xsl:value-of select="name"/>,</xsl:template>
```

```
            <xsl:value-of select="year"/>,</xsl:stylesheet>
```

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

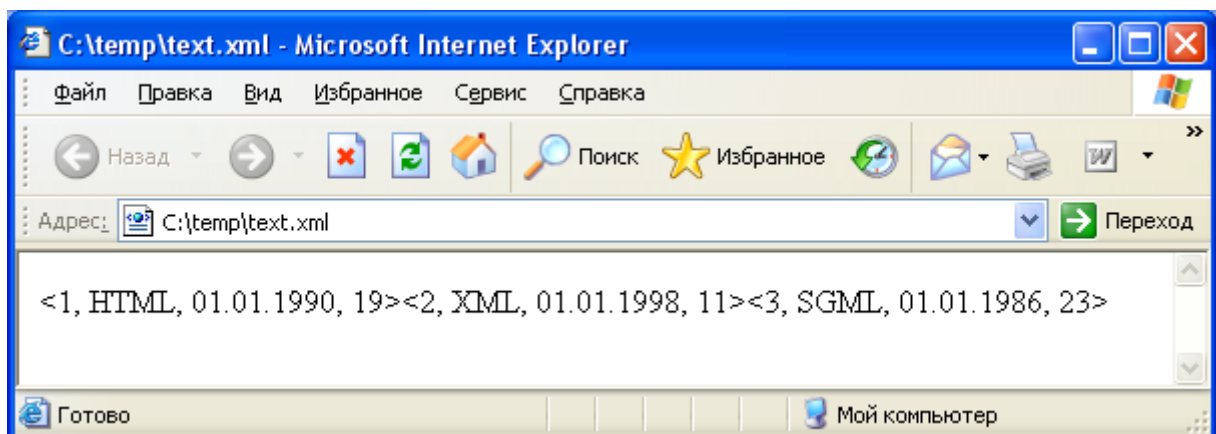
```
            <xsl:text>&gt;</xsl:text>
```

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

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

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

## Просмотр результата в браузере:



[Оглавление](#)

Рис. 2.23. Результат выполнения примера 2.10.

Формирование текстового документа может быть полезно, например, в том случае, когда необходимо сгенерировать на основе XML-документа программу на каком-либо языке программирования. Например, можно выгрузить данные из БД в формате XML, и с помощью XSLT-преобразования сгенерировать SQL-сценарий на вставку данных в другую БД.

### 2.2.11 Копирование узлов

Копирование узлов удобно использовать в том случае, когда элементы входного документа нужно поместить в выходной документ без изменений.

Для копирования узлов применяют команду

```
<xsl:copy> узлы для копирования </xsl:copy>
```

или команду

```
<xsl:copy-of select="XPath-выражение, задающее узлы для  
копирования"/>
```

В следующем примере HTML-фрагменты из элементов text преобразуются в элементы P (параграф текста).

#### Пример 2.11.

##### Файл XML:

```
<?xml version="1.0" encoding="Windows-1251"?>
<?xml-stylesheet type="text/xsl" href="copy_of.xsl"?>
<texts>
  <text>Обычный текст</text>
  <text><b>Полужирный текст</b></text>
  <text><i>Курсив</i></text>
</texts>
```

##### Файл XSLT:

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

```

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

<xsl:template match="/">

    <xsl:for-each select="texts/text">
        <P>
            <xsl:copy-of select="* | text()" />
        </P>
    </xsl:for-each>

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

```

### Просмотр результата в браузере:

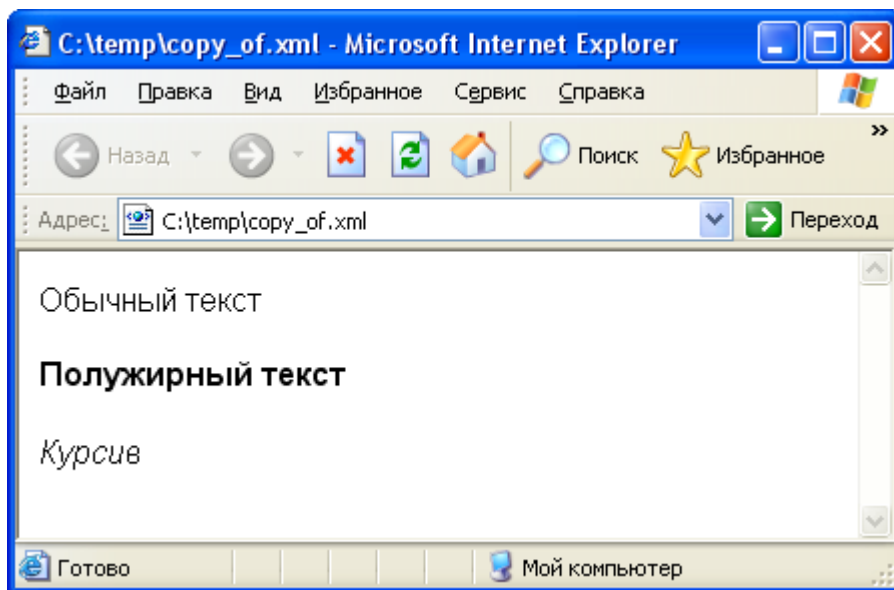


Рис. 2.24. Результат выполнения примера 2.11.

Обратите внимание на элемент

```
<xsl:copy-of select="* | text()" />
```

XPath-выражение в атрибуте `select` выбирает узлы элементов «\*» или текстовые узлы «`text()`». Благодаря «\*» в выходной документ копируются элементы «`<b>Полужирный текст</b>`» и «`<i>Курсив</i>`», а благодаря «`text()`», копируется текст «Обычный текст».

[Оглавление](#)

XPath-выражения с использованием оператора объединения множеств « | » часто применяются при копировании узлов.

### 2.2.12 Расширения XSLT

Расширения XSLT (XSLT extensions) – это технология, которая позволяет вызывать из XSLT-преобразования исполняемый код.

Расширения реализуются различным образом в разных XSLT-процессорах. Единого стандарта на расширения не существует. Но почти во всех процессорах вызов расширений производится с использованием пространств имен, как в следующем примере.

Языки, с помощью которых расширяются XSLT-процессоры, могут быть различными. Если XSLT-процессор разработан на платформе Java, как правило, он расширяется с использованием Java-классов.

XSLT-процессор, разработанный Microsoft, может быть расширен с помощью языков сценариев (например, JavaScript) или с помощью языков, применяемых в .NET-платформе.

В следующем примере в XSLT-преобразовании вызывается функция на языке JavaScript.

#### Пример 2.12.

##### Файл XSLT:

```
<?xml version="1.0" encoding="Windows-1251"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt"
  xmlns:user="http://ns1.org">

<xsl:template match="/">
  <HTML>
  <BODY>
  <TABLE BORDER="2">
```

```

<TR>
  <TH>№</TH>
  <TH>Язык разметки</TH>
  <TH>Год создания</TH>
  <TH>Возраст технологии (лет)</TH>
</TR>
<xsl:for-each select="languages/language">
  <TR>
    <TD><xsl:value-of select="@id"/></TD>
    <TD><xsl:value-of select="name"/></TD>
    <TD><xsl:value-of select="year"/></TD>

    <!-- Вызов функции в процессе преобразования -->
    <!-- С помощью функции XPath "number" содержимое элемента
    преобразуется в число -->
    <TD>
      <xsl:value-of select="user:count_howold(number(howold))"/>
    </TD>

  </TR>
</xsl:for-each>
</TABLE>
</BODY>
</HTML>
</xsl:template>

<msxsl:script language="JavaScript" implements-prefix="user">
//Функция возвращает значение, умноженное на 3
function count_howold(howold_param)
{
  var res = howold_param * 3;
  return res;
}
</msxsl:script>
</xsl:stylesheet>

```

## Просмотр результата в браузере:

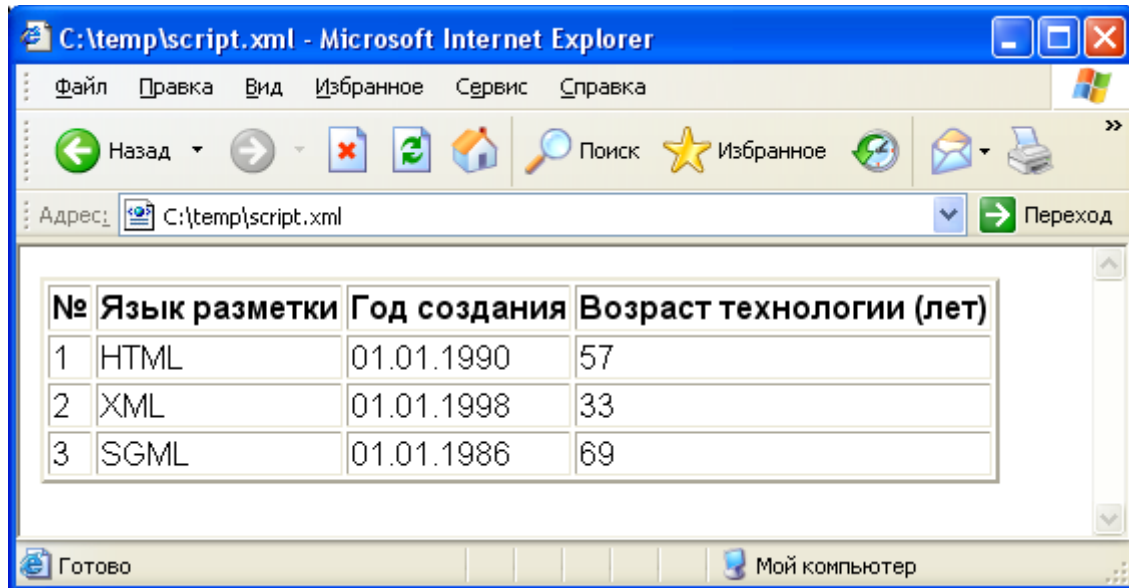


Рис. 2.25. Результат выполнения примера 2.12.

Поскольку расширение использует особенности XSLT-процессора Microsoft, то оно будет работать только под Internet Explorer.

Рассмотрим более подробно, как вызывается расширение.

```
<?xml version="1.0" encoding="Windows-1251"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt"
  xmlns:user="http://ns1.org">
```

В элементе `stylesheet` кроме стандартного для XSLT пространства имен с префиксом `xsl`, объявлено пространство имен с префиксом `msxsl` для вызова расширений на JavaScript.

Также объявлено пространство имен с префиксом `user` для вызова пользовательских функций. Название `"http://ns1.org"` может быть произвольным.

```
<xsl:value-of select="user:count_howold(number(howold))"/>
```

В XPath-выражении производится вызов функции `count_howold`, написанной на JavaScript. Префикс «`user:`» указывает пространство имен, к которому относится эта функция. С помощью функции XPath «`number`» содержимое

элемента `howold` преобразуется в число, это значение передается функции в качестве параметра.

```
<msxsl:script language="JavaScript" implements-prefix="user">
//Функция возвращает значение, умноженное на 3
function count_howold(howold_param)
{
    var res = howold_param * 3;
    return res;
}
</msxsl:script>
```

Элемент `msxsl:script` используется для создания расширения на языках сценариев в XSLT-процессоре Microsoft.

Атрибут `language="JavaScript"` определяет язык сценария.

Атрибут `implements-prefix="user"` определяет префикс пространства имен, через который вызываются пользовательские функции. В нашем случае вызов функции производится через префикс «user:»

```
<xsl:value-of select="user:count_howold(number(howold))"/>
```

Обратите внимание, что комментарий

```
//Функция возвращает значение, умноженное на 3
```

может быть записан не по правилам XML, а по правилам JavaScript, так как он находится внутри `msxsl:script`.

Функция `count_howold`, написанная на JavaScript, возвращает параметр, умноженный на три.

С использованием расширений XSLT могут быть решены две нестандартные задачи:

1. XSLT можно превратить из средства преобразования данных в средство загрузки данных. Например, если написать расширение, которое будет



осуществлять вставку данных в БД, то данные можно не преобразовывать в другой формат, а загружать в БД.

2. Преобразование данных из XML в двоичный формат. Например, если написать расширение, которое будет добавлять данные в двоичный файл, то данные можно преобразовывать из XML в двоичный формат.

### 2.2.13 Использование ключей поиска

Ключи поиска в XSLT напоминают индексы в реляционной БД. Они предназначены для поиска фрагментов документов по значению ключа.

Ключ объявляется с помощью элемента XSLT

```
<xsl:key name="название ключа" match="индексируемый узел" use="часть узла, которая является ключом" />
```

Для поиска по ключу используется функция языка XPath

```
key("название ключа", "значение ключа, соответствующее атрибуту use")
```

Функция key возвращает множество узлов указанных в match, которые соответствуют заданному значению ключа.

Пример использования ключей:

#### Пример 2.13.

##### Файл XML:

```
<?xml version="1.0" encoding="Windows-1251"?>
<?xml-stylesheet type="text/xsl" href="key.xsl"?>
<titles>
  <book title="Книга про XML" author="Иванов"/>
  <book title="Книга про HTML" author="Иванов"/>
  <book title="Другая книга про XML" author="Петров"/>
  <book title="Другая книга про HTML" author="Петров"/>
</titles>
```

##### Файл XSLT:

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

```

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

<xsl:key name="title-search" match="book" use="@author"/>

<xsl:template match="/">
  <HTML>
    <BODY>
      <P><B>Книги автора Иванова:</B></P>
      <xsl:for-each select="key('title-search', 'Иванов')">
        <P>
          <xsl:number level="single" value="position()" format="1"/>.
          <xsl:value-of select="@title"/>
        </P>
      </xsl:for-each>

      <P><B>Книги автора Петрова:</B></P>
      <xsl:for-each select="key('title-search', 'Петров')">
        <P>
          <xsl:number level="single" value="position()" format="I"/>.
          <xsl:value-of select="@title"/>
        </P>
      </xsl:for-each>
    </BODY>
  </HTML>
</xsl:template>
</xsl:stylesheet>

```

### **Просмотр результата в браузере:**

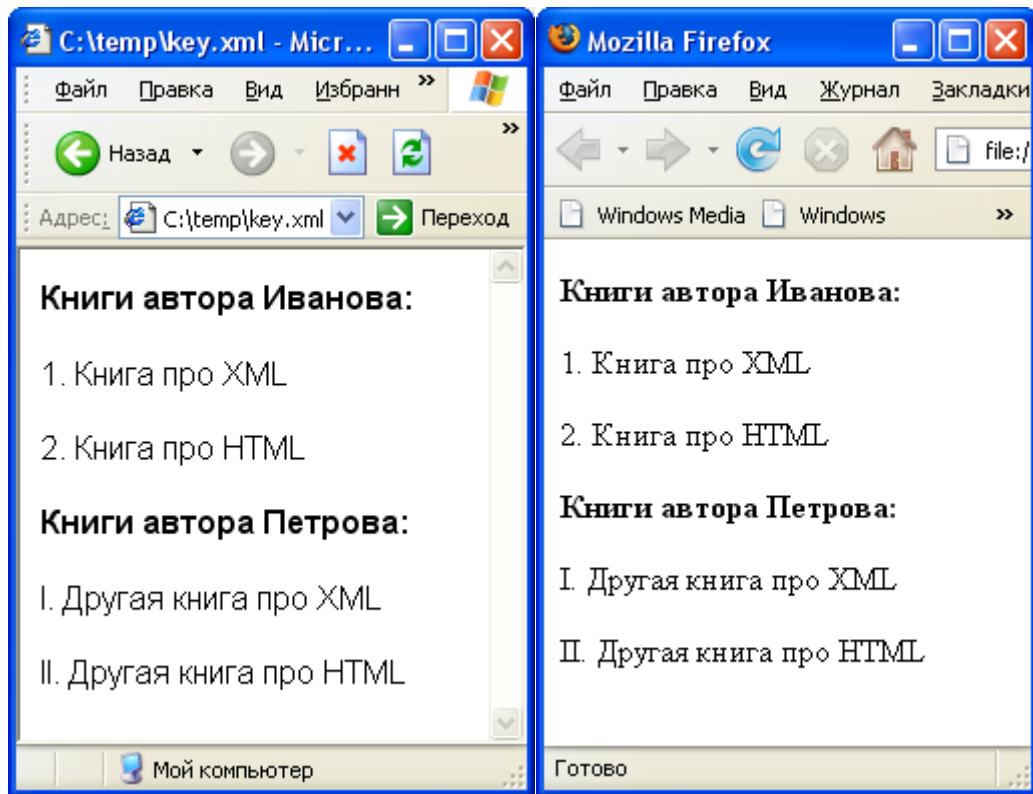


Рис. 2.26. Результат выполнения примера 2.13.

Рассмотрим более подробно текст преобразования XSLT.

```
<?xml version="1.0" encoding="Windows-1251"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:key name="title-search" match="book" use="@author"/>
```

Объявление ключа с названием title-search. Индексируется элемент документа book, функция key будет возвращать именно эти элементы. Ключом является атрибут author элемента book.

Обратите внимание, что XPath-выражение в атрибуте use вычисляется внутри контекста, указанного в атрибуте match.

```
<xsl:template match="/">
  <HTML>
    <BODY>
      <P><B>Книги автора Иванова:</B></P>
      <xsl:for-each select="key('title-search', 'Иванов')">
```

[Оглавление](#)

Функция `key` вызывается для ключа `'title-search'`, значением ключа является `'Иванов'`. Функция вернет множество элементов `book`, у которых значение атрибута `author='Иванов'`.

```
<P>
```

```
<xsl:number level="single" value="position()" format="1"/>.
```

Элемент `number` позволяет генерировать номера в выходном документе. В этом примере генерируется одноуровневая нумерация (`level="single"`), номер элемента соответствует текущей позиции в цикле (`value="position()"`). Значение атрибута `format="1" | "I"`, соответственно нумерация арабскими или римскими цифрами.

Элемент `number` также позволяет реализовывать многоуровневую нумерацию.

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

Так как функция `key` возвращает элементы `book`, они перебираются в цикле `for-each`, то XPath-выражения в теле цикла должны строиться в контексте `book`. XPath-выражение `@title` возвращает атрибут `title` текущего элемента `book`.

```
</P>
```

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

Аналогично выводятся книги второго автора, функция `key` используется с другим значением ключа поиска

```
<P><B>Книги автора Петрова:</B></P>
```

```
<xsl:for-each select="key('title-search', 'Петров')">
```

```
<P>
```

```
<xsl:number level="single" value="position()" format="I"/>.
```

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

```
</P>
```

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

```
</BODY>
```

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

### 2.2.14 Обработка документа из нескольких секций (использование переменных)

Как было показано ранее в примере использования сортировки, для создания переменных используется элемент `variable`. Элемент `variable` может быть использован в виде:

```
<xsl:variable name="название_переменной" select="значение"/>
```

или

```
<xsl:variable name="название_переменной">значение</xsl:variable>
```

Для обращения к переменным перед названием ставится знак \$, например:

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

По аналогии с функциональными языками программирования переменные в XSLT не могут изменять значение после присваивания (*immutable variable*).

Фактически они являются константами, присвоить значения которым можно только один раз. Наиболее часто переменные используют для удобства разработки шаблонов, занося в них промежуточные фрагменты документов.

В качестве примера использования переменных рассмотрим модификацию примера с формированием таблицы. Особенность состоит в том, что информация о языках разметки находится в нескольких секциях XML-документа. Для удобства доступа к секциям используются переменные.

#### Пример 2.14.

**Файл XML (в данном примере тестовое содержимое элементов `description` содержит фрагменты описаний Википедии, посвященных соответствующим языкам разметки):**

```
<?xml version="1.0" encoding="Windows-1251"?>
<?xml-stylesheet type="text/xsl" href="Example_var.xsl"?>
<!-- Языки разметки -->
```

```

<root>
  <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>
  <descriptions>
    <description language="HTML">
      HTML (от англ. HyperText Markup Language – «язык
      гипертекстовой разметки») – стандартный язык разметки документов во
      Всемирной паутине. Большинство веб-страниц содержат описание
      разметки на языке HTML (или XHTML). Язык HTML интерпретируется
      браузерами и отображается в виде документа в удобной для человека
      форме.
    </description>
    <description language="XML">
      XML (англ. eXtensible Markup Language – расширяемый язык
      разметки; произносится [экс-эм-эл]) – рекомендованный Консорциумом
      Всемирной паутины (W3C) язык разметки. Спецификация XML описывает
      XML-документы и частично описывает поведение XML-процессоров
      (программ, читающих XML-документы и обеспечивающих доступ к их
      содержимому).
    </description>
  </descriptions>

```

```
<description language="SGML">
```

SGML (англ. Standard Generalized Markup Language – стандартный обобщённый язык разметки; произносится [эс-джи-эм-эл]) – метаязык, на котором можно определять язык разметки для документов. SGML – наследник разработанного в 1969 году в IBM языка GML (Generalized Markup Language).

```
</description>
```

```
</descriptions>
```

```
</root>
```

## 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>Описание</TH>
```

```
</TR>
```

```
<xsl:for-each select="//language">
```

```
<!-- Перебор в цикле всех элементов language, вложенных в элемент
languages. -->
```

```
<xsl:variable name="name_text" select="name/text()"/>
```

```
<!-- Создание переменной содержащей текстовое значение элемента
name. -->
```

```
<xsl:variable name="descr"
```

```
select="//description[@language=$name_text]"/>
```

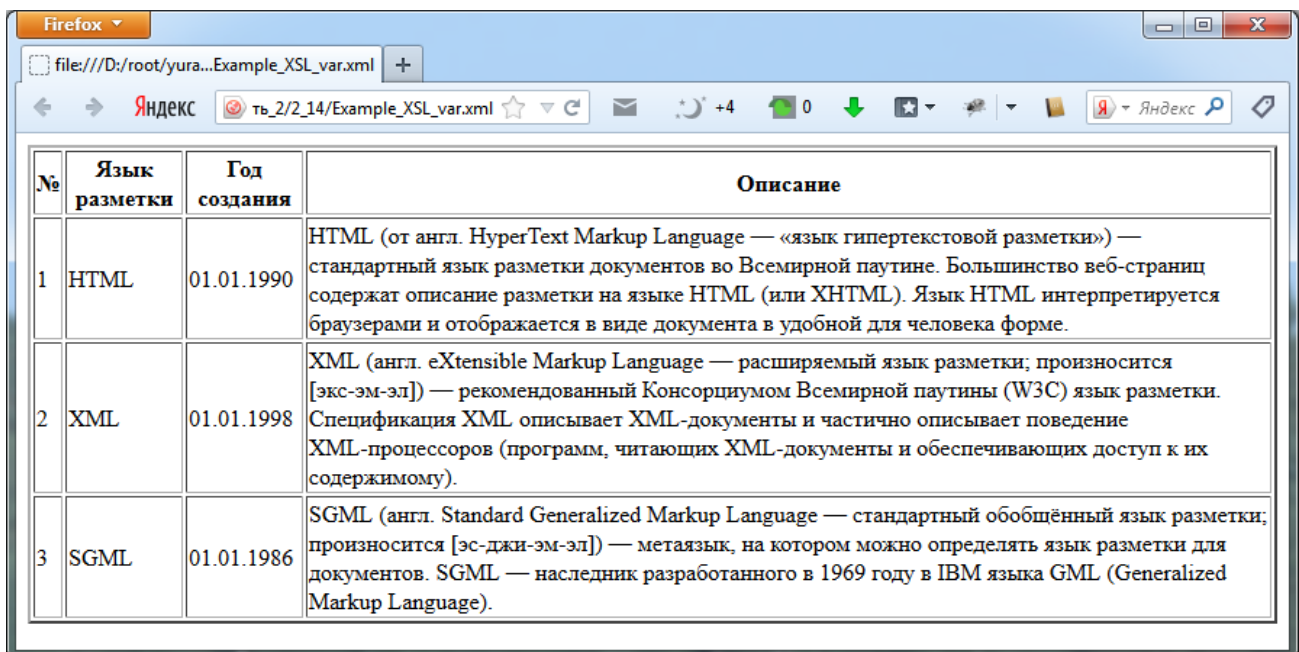
[Оглавление](#)

```

<!-- Создание переменной для доступа к другой секции. -->
<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="$descr"/></TD>
<!-- Выбор описания из переменной -->
</TR>
</xsl:for-each>
</TABLE>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>

```

### Просмотр результата в браузере:



№	Язык разметки	Год создания	Описание
1	HTML	01.01.1990	HTML (от англ. HyperText Markup Language — «язык гипертекстовой разметки») — стандартный язык разметки документов во Всемирной паутине. Большинство веб-страниц содержат описание разметки на языке HTML (или XHTML). Язык HTML интерпретируется браузерами и отображается в виде документа в удобной для человека форме.
2	XML	01.01.1998	XML (англ. eXtensible Markup Language — расширяемый язык разметки; произносится [экс-эм-эл]) — рекомендованный Консорциумом Всемирной паутины (W3C) язык разметки. Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому).
3	SGML	01.01.1986	SGML (англ. Standard Generalized Markup Language — стандартный обобщённый язык разметки; произносится [эс-джи-эм-эл]) — метаязык, на котором можно определять язык разметки для документов. SGML — наследник разработанного в 1969 году в IBM языка GML (Generalized Markup Language).

Рис. 2.27. Результат выполнения примера 2.14.

В данном примере в XSLT-преобразовании создается две переменных.



```
<xsl:variable name="name_text" select="name/text()" />
```

Создание переменной содержащей текстовое значение элемента name. «name/text()» – из элемента name выбирается вложенный текстовый узел.

```
<xsl:variable name="descr"
select="//description[@language=$name_text]" />
```

Создание переменной, в которую заносится описание языка разметки (элемент description) из другой секции документа. Атрибут language искомого элемента description должен совпадать со значением предварительно созданной переменной name\_text.

## 2.2.15 Создание кросс-таблицы

Рассмотрим более сложный пример создания кросс-таблицы. Этот пример не содержит новых команд XSLT, по сравнению с предыдущими примерами.

Сложность создания кросс-таблицы состоит в том, что заголовки строк и столбцов формируются динамически на основе данных, также динамически формируются значения ячеек.

Еще одна сложность состоит в том, что необходимо использовать группировку данных, которая не реализована в XSLT 1.0 явным образом.

### Пример 2.15.

#### Файл XML:

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="crosstable.xsl"?>
<Institute>

  <Data>
    <Department>ИУ - 5</Department>
    <Student>Иванов И.И.</Student>
    <CourseName>Интернет-технологии</CourseName>
    <Result>5</Result>
  </Data>
```

```
<Data>
  <Department>ИУ - 5</Department>
  <Student>Иванов И.И.</Student>
  <CourseName>Язык XML</CourseName>
  <Result>5</Result>
</Data>
```

```
<Data>
  <Department>ИУ - 7</Department>
  <Student>Петров П.П.</Student>
  <CourseName>Интернет-технологии</CourseName>
  <Result>5</Result>
</Data>
```

```
<Data>
  <Department>ИУ - 5</Department>
  <Student>Сепреев С.С.</Student>
  <CourseName>Язык XML</CourseName>
  <Result>5</Result>
</Data>
```

```
</Institute>
```

Файл с данными состоит из нескольких элементов Data с вложенными элементами. Элемент Department содержит название кафедры, элемент Student – имя студента, элемент CourseName – название курса, элемент Result – оценку.

Необходимо в строках таблицы выдать список студентов, сгруппированный по кафедрам, в заголовках столбцов выдать список курсов. Если у студента есть оценка за курс, то она выводится на пересечении строки и столбца. При этом не у всех студентов обязательно есть оценки по всем курсам.

### Файл XSLT:

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

```

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

<!-- Объявление ключей для группировки и поиска -->
<xsl:key name="CourseName" match="Data" use="CourseName"/>
<xsl:key name="Department" match="Data" use="Department"/>
<xsl:key name="Student" match="Data" use="concat(Department,
Student)"/>
<xsl:key name="Res" match="Data" use="concat(Department, Student,
CourseName)"/>

<xsl:template match="/">
<!-- Правило обработки корневого элемента XML - документа -->
  <HTML>
    <BODY>
      <TABLE BORDER="2">
        <TR>
          <!-- Пустая ячейка соответствует первому столбцу -->
          <TH/>
          <xsl:for-each select="//Data[generate-id(.)=generate-
id(key('CourseName',CourseName))]">
            <xsl:sort select="CourseName" data-type="text"/>
            <TH><xsl:value-of select="CourseName"/></TH>
          </xsl:for-each>
        </TR>
        <xsl:for-each select="//Data[generate-id(.)=generate-
id(key('Department',Department))]">
          <xsl:sort select="Department" data-type="text"/>
          <!-- Данные о кафедре -->
          <xsl:variable name="var1" select="Department"/>
          <TR>
            <TH><xsl:value-of select="$var1"/></TH>
          </TR>
          <!-- Данные о студенте -->

```

```

        <xsl:for-each select="//Data[generate-id(.)=generate-
id(key('Student',concat($var1, Student)))]">
            <xsl:sort select="Student" data-type="text"/>
            <xsl:variable name="var2" select="Student"/>
            <TR>
                <TD><xsl:value-of select="$var2"/></TD>
                <!-- Перебор предметов -->
                <xsl:for-each select="//Data[generate-id(.)=generate-
id(key('CourseName',CourseName))]">
                    <xsl:sort select="CourseName" data-type="text"/>
                    <xsl:variable name="var3" select="CourseName"/>
                    <!-- Получение по ключу элемента Data, который соответствует текущей
комбинации кафедры, студента и предмета -->
                    <xsl:variable name="res" select="key('Res',concat($var1, $var2,
$var3))"/>
                    <TD><xsl:value-of select="$res/Result"/></TD>
                </xsl:for-each>
            </TR>
        </xsl:for-each>
    </xsl:for-each>
</TABLE>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>

```

### Просмотр результата в браузере:

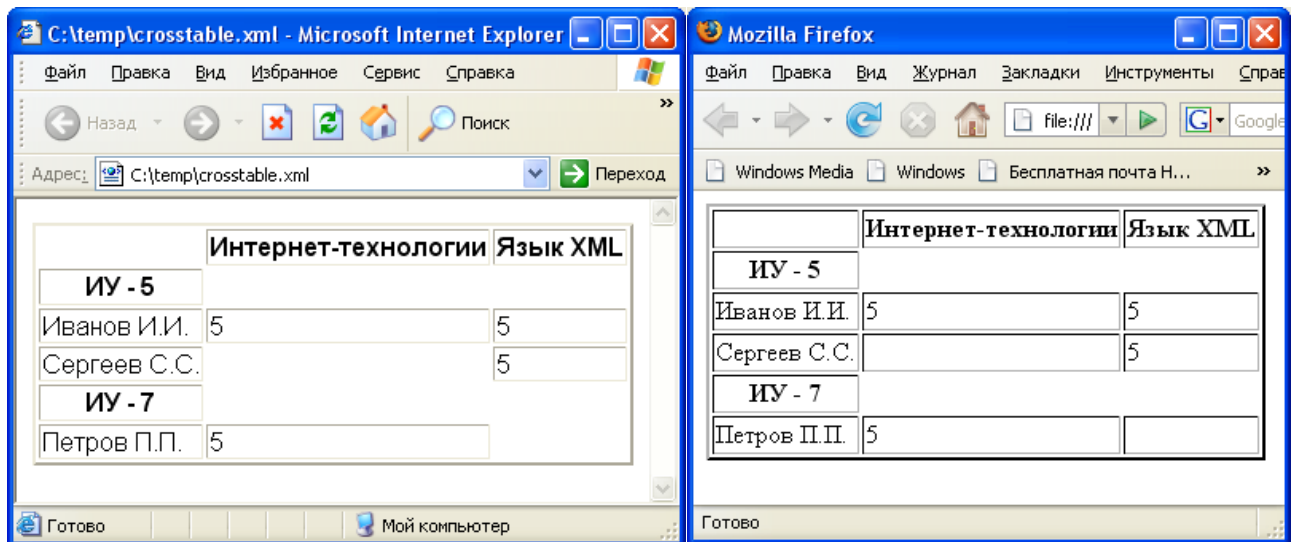


Рис. 2.28. Результат выполнения примера 2.15.

Рассмотрим более подробно текст преобразования XSLT.

```
<?xml version="1.0" encoding="Windows-1251"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<!-- Объявление ключей для группировки и поиска -->
<xsl:key name="CourseName" match="Data" use="CourseName"/>
```

Ключ используется для группировки данных по курсам.

```
<xsl:key name="Department" match="Data" use="Department"/>
```

Ключ используется для группировки данных по кафедрам.

```
<xsl:key name="Student" match="Data" use="concat(Department,
Student)"/>
```

Ключ используется для группировки данных по студентам внутри кафедры.

```
<xsl:key name="Res" match="Data" use="concat(Department, Student,
CourseName)"/>
```

Ключ используется для получения оценки по кафедре, студенту и курсу.

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

```

<!-- Правило обработки корневого элемента XML - документа -->
<HTML>
<BODY>
<TABLE BORDER="2">
<TR>
<!-- Пустая ячейка соответствует первому столбцу -->
<TH/>
<!-- В столбцах отображаются названия предметов, по предметам
производится группировка -->
<xsl:for-each select="//Data[generate-id(.)=generate-
id(key('CourseName',CourseName))]">

```

XPath-выражение в атрибуте `select` выбирает список курсов с группировкой (с исключением дубликатов).

Такой способ группировки в XSLT называется «методом Мюнха» (по имени автора метода). Более подробно этот метод описан в [Валиков, 2002].

Функция `generate-id(элемент документа)` генерирует уникальный идентификатор для элемента документа. В соответствии со стандартом, эта функция является детерминированной, то есть для одного и того же элемента документа (параметра функции) она всегда возвращает одинаковое значение.

Если параметром функции `generate-id` является множество узлов, то она возвращает идентификатор для первого узла. То есть «`generate-id(key('CourseName',CourseName))`» вернет уникальный идентификатор первого элемента `Data` из тех, которые вернет функция `key`.

Не важно, какой именно элемент `Data` будет стоять первым. Важно то, что это выражение для одного и того же параметра `CourseName` будет возвращать один и тот же уникальный идентификатор, который соответствует какому-то элементу `Data` из ключа. Фактически будет возвращаться идентификатор группы.

Выражение «`generate-id(.)`» возвращает уникальный идентификатор для текущего элемента.

Выражение `//Data[generate-id(.)=generate-id(key('CourseName',CourseName))]` проверяет, что текущий элемент `Data` является первым элементом в группе.

Параметр `CourseName`, который передается в функцию `key`, вложен в текущий элемент «`.`», фактически это «`./CourseName`». То есть левая и правая часть оператора равенства связаны по значению элемента `CourseName`.

Таким образом, из всех элементов `Data` с одинаковым значением вложенного элемента `CourseName` возвращается только один, который соответствует первому элементу `Data`, возвращаемому функцией `key`. То есть реализуется группировка (исключение дубликатов) по элементу `CourseName`.

В случае многоуровневой группировки в параметр ключа включаются все уровни группировки путем конкатенации строк, например `concat(Department, Student)`

В версии стандарта XSLT 2.0 этот метод можно не использовать, так как предусмотрен специальный элемент `xsl:for-each-group`, который реализует группировку.

```
<xsl:sort select="CourseName" data-type="text"/>
```

Список курсов сортируется по возрастанию.

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

В ячейку заголовка столбца таблицы заносится название курса.

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

```
</TR>
```

```
<xsl:for-each select="//Data[generate-id(.)=generate-  
id(key('Department', Department))]">
```

Группировка по кафедрам с использованием рассмотренного метода группировки.

```
<xsl:sort select="Department" data-type="text"/>
```

Сортировка по названию кафедры.

```
<!-- Данные о кафедре -->
```

```
<xsl:variable name="var1" select="Department"/>
```

Создание переменной с названием кафедры

```
<TR>
```

```
<TH><xsl:value-of select="$var1"/></TH>
```

В ячейку заголовка строки таблицы заносится название кафедры. В строки с названием кафедр данные не выводятся.

```
</TR>
```

```
<!-- Данные о студенте -->
```

```
<xsl:for-each select="//Data[generate-id(.)=generate-id(key('Student',concat($var1, Student)))]">
```

Группировка по именам студентов внутри кафедры.

```
<xsl:sort select="Student" data-type="text"/>
```

```
<xsl:variable name="var2" select="Student"/>
```

```
<TR>
```

```
<TD><xsl:value-of select="$var2"/></TD>
```

```
<xsl:for-each select="//Data[generate-id(.)=generate-id(key('CourseName',CourseName)))]">
```

Для каждого студента необходимо осуществить перебор всех предметов, так как название предмета используется в ключе поиска.

```
<xsl:sort select="CourseName" data-type="text"/>
```

```
<xsl:variable name="var3" select="CourseName"/>
```

```
<xsl:variable name="res" select="key('Res',concat($var1, $var2, $var3))"/>
```

Получение по ключу Res элемента Data, который соответствует текущей комбинации кафедры, студента и предмета.

```
<TD><xsl:value-of select="$res/Result"/></TD>
```



Вывод оценки Result из полученного по ключу Res элемента Data, который соответствует текущей комбинации кафедры, студента и предмета. Если такой элемент не обнаружен, то `xsl:value-of` возвращает пустое значение.

```

        </xsl:for-each>
    </TR>
</xsl:for-each>
</xsl:for-each>
</TABLE>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>

```

## 2.3 Технология XSLT 2.0

В предыдущем разделе рассмотрены основы XSLT 1.0 [XSLT, 1999]. В январе 2007 года появился стандарт XSLT 2.0 [XSLT, 2007].

Основным отличием второй версии является то, что в нем вместо XPath 1.0 используется XPath 2.0. Также важными дополнениями в этом стандарте являются возможность использования группировки и создания пользовательских функций.

Далее мы рассмотрим данные особенности на конкретных примерах.

### 2.3.1 Инструментальные средства для работы с XSLT 2.0

В отличие от XSLT 1.0, XSLT 2.0 не поддерживается в большинстве браузеров непосредственно. Для работы с XSLT 2.0 требуются дополнительные инструментальные средства.

Наиболее известным XSLT 2.0 – процессором с открытым исходным кодом является Saxon (<http://saxon.sourceforge.net/>). В частности он поставляется в виде решения на JavaScript – Saxon CE (<http://www.saxonica.com/ce/index.xml>), которое можно запустить в браузере без дополнительных библиотек.

Для дальнейших примеров мы будем использовать средство exselt, которое поддерживает стандарты XSLT 2.0 и XSLT 3.0. Данный продукт не является средством с открытым исходным кодом, но включает демонстрационный сайт, который позволяет осуществлять преобразования.

Для запуска оставшейся части примеров в данной главе необходимо открыть демонстрационный сайт <http://exselt.net/demo>, скопировать файлы XML и XSL в соответствующие окна и нажать кнопку «Generate».

### 2.3.2 Использование группировки

Для группировки используется элемент `xsl:for-each-group`. Этот элемент содержит атрибут `select` (как в элементе `xsl:for-each`) который используется для выбора элементов, а также атрибут `group-by` который содержит выражение группировки.

Использование элемента `xsl:for-each-group` рассмотрим на примере формирования кросс-таблицы. С использованием возможностей XSLT 2.0 данный пример реализуется значительно проще.

#### Пример 2.16.

##### Файл XML:

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

  <Data>
    <Department>ИУ - 5</Department>
    <Student>Иванов И.И.</Student>
    <CourseName>Интернет-технологии</CourseName>
    <Result>5</Result>
  </Data>

  <Data>
    <Department>ИУ - 5</Department>
    <Student>Иванов И.И.</Student>
    <CourseName>Язык XML</CourseName>
```

```
<Result>5</Result>
</Data>
```

```
<Data>
  <Department>ИУ - 7</Department>
  <Student>Петров П.П.</Student>
  <CourseName>Интернет-технологии</CourseName>
  <Result>5</Result>
</Data>
```

```
<Data>
  <Department>ИУ - 5</Department>
  <Student>Сегреев С.С.</Student>
  <CourseName>Язык XML</CourseName>
  <Result>5</Result>
</Data>
```

```
</Institute>
```

Файл с данными состоит из нескольких элементов Data с вложенными элементами. Элемент Department содержит название кафедры, элемент Student – имя студента, элемент CourseName – название курса, элемент Result – оценку.

Необходимо в строках таблицы выдать список студентов, сгруппированный по кафедрам, в заголовках столбцов выдать список курсов. Если у студента есть оценка за курс, то она выводится на пересечении строки и столбца. При этом не у всех студентов обязательно есть оценки по всем курсам.

### Файл XSLT:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0">
<xsl:template match="/">
<!-- Правило обработки корневого элемента XML - документа -->
<HTML>
```

```

<BODY>
<TABLE BORDER="2">
  <TR>
    <!-- Пустая ячейка соответствует первому столбцу -->
    <TH/>
    <!-- В столбцах отображаются названия предметов, по предметам
производится группировка -->
    <xsl:for-each-group select="//Data" group-by="CourseName">
      <xsl:sort select="CourseName" data-type="text"/>
      <TH><xsl:value-of select="CourseName"/></TH>
    </xsl:for-each-group>
  </TR>
  <xsl:for-each-group select="//Data" group-by="Department">
    <xsl:sort select="Department" data-type="text"/>
    <!-- Данные о кафедре -->
    <xsl:variable name="var1" select="Department/text()"/>
    <TR>
      <TH><xsl:value-of select="$var1"/></TH>
    </TR>
    <!-- Данные о студенте -->
    <xsl:for-each-group select="//Data[Department=$var1]" group-
by="Student">
      <xsl:sort select="Student" data-type="text"/>
      <xsl:variable name="var2" select="Student/text()"/>
      <TR>
        <TD><xsl:value-of select="$var2"/></TD>
        <!-- Перебор предметов -->
        <xsl:for-each-group select="//Data" group-
by="CourseName">
          <xsl:sort select="CourseName" data-type="text"/>
          <xsl:variable name="var3" select="CourseName/text()"/>
          <xsl:variable name="res"
select="//Data[Department=$var1 and Student=$var2 and
CourseName=$var3]"/>
          <TD>

```

```

        <xsl:value-of select="$res/Result"/>
    </TD>
</xsl:for-each-group>
</TR>
</xsl:for-each-group>
</xsl:for-each-group>
</TABLE>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>

```

### Просмотр результата в браузере:

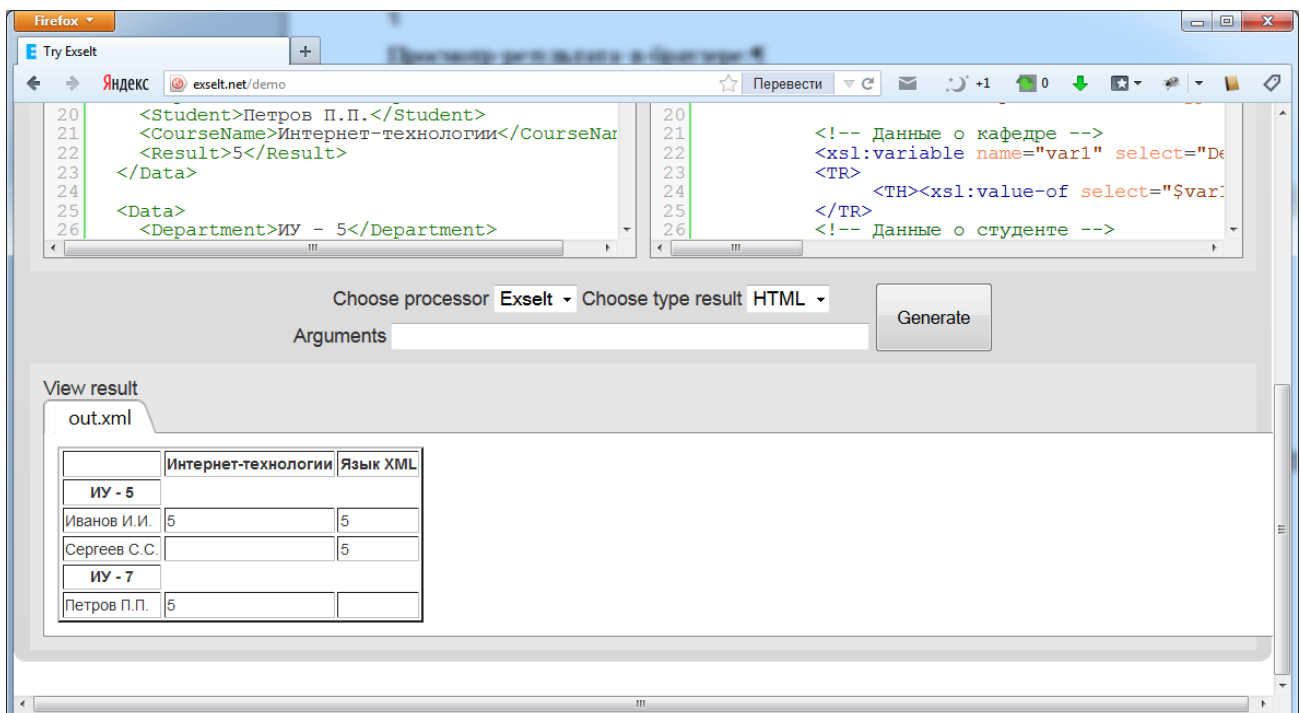


Рис. 2.29. Результат выполнения примера 2.16.

Рассмотрим более подробно использование группировки. Вместо метода Мюнха, который предполагал использование ключей и был достаточно громоздким, в XSLT 2.0 используется элемент `xsl:for-each-group`.

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0">

```

```

<xsl:template match="/">
<!-- Правило обработки корневого элемента XML - документа -->
  <HTML>
  <BODY>
  <TABLE BORDER="2">
    <TR>
      <!-- Пустая ячейка соответствует первому столбцу -->
      <TH/>

```

**Вывод заголовочной части HTML-документа.**

```

      <!-- В столбцах отображаются названия предметов, по предметам
      производится группировка -->
      <xsl:for-each-group select="//Data" group-by="CourseName">
        <xsl:sort select="CourseName" data-type="text"/>
        <TH><xsl:value-of select="CourseName"/></TH>
      </xsl:for-each-group>

```

В данном фрагменте выбираются все элементы Data в документе. Выбранные элементы группируются на основании значения элемента CourseName, вложенного в элемент Data. Далее производится сортировка элементов CourseName с помощью xsl:sort и вывод с помощью xsl:value-of.

```

    </TR>
    <xsl:for-each-group select="//Data" group-by="Department">
      <xsl:sort select="Department" data-type="text"/>

```

**Группировка и сортировка элементов Data по кафедрам.**

```

      <!-- Данные о кафедре -->
      <xsl:variable name="var1" select="Department/text()" />

```

Переменная var1 содержит название кафедры (текстовое значение элемента Department).

```

    <TR>
      <TH><xsl:value-of select="$var1"/></TH>

```

В ячейку заголовка строки таблицы заносится название кафедры. В строки с названием кафедр данные не выводятся.

```
</TR>
<!-- Данные о студенте -->
<xsl:for-each-group select="//Data[Department=$var1]" group-
by="Student">
  <xsl:sort select="Student" data-type="text"/>
```

Группировка и сортировка по именам студентов внутри кафедры. В атрибуте select элемента xsl:for-each-group выбираются только студенты по текущей кафедре (перебор кафедр осуществляется в цикле верхнего уровня).

```
<xsl:variable name="var2" select="Student/text()"/>
<TR>
  <TD><xsl:value-of select="$var2"/></TD>
```

Создание переменной var2 для имени студента и вывод имени студента.

```
<!-- Перебор предметов -->
<xsl:for-each-group select="//Data" group-
by="CourseName">
  <xsl:sort select="CourseName" data-type="text"/>
```

Для каждого студента осуществляется перебор всех предметов.

```
<xsl:variable name="var3" select="CourseName/text()"/>
```

Создание переменной var3 для названия предмета.

```
<xsl:variable name="res"
select="//Data[Department=$var1 and Student=$var2 and
CourseName=$var3]"/>
```

Поиск элемента Data, который соответствует текущим кафедре, студенту и предмету. Такой элемент не обязательно должен присутствовать в документе.

```
<TD>
```

```
<xsl:value-of select="$res/Result"/>
```

Вывод оценки Result из найденного элемента Data, который соответствует текущей комбинации кафедры, студента и предмета. Если такой элемент не обнаружен, то xsl:value-of возвращает пустое значение.

```

        </TD>
    </xsl:for-each-group>
</TR>
</xsl:for-each-group>
</xsl:for-each-group>
</TABLE>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>

```

### 2.3.3 Пользовательские функции

Пользовательские функции (stylesheet function) создаются с помощью элемента xsl:function. Такие функции могут быть вызваны из XPath-выражений.

Пример использования функции является модификацией примера формирования таблицы.

#### Пример 2.17.

##### Файл XML:

```

<?xml version="1.0" encoding="utf-8"?>
<!-- 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="utf-8"?>
<!--XSLT - документ является XML - документом. -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:str="http://example.com/namespace" version="2.0">
<!-- 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>
    </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="str:KeyValue(name, year)"/></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>
    </TR>
</xsl:for-each>
</TABLE>
</BODY>
</HTML>
</xsl:template>

<!-- Функция вывода в виде ключа и значения -->
<xsl:function name="str:KeyValue" as="xs:string">
    <xsl:param name="key" as="xs:string"/>
    <xsl:param name="value" as="xs:string"/>
    <p>
        <xsl:value-of select="$key" />
        ->
        <xsl:value-of select="$value" />
    </p>
</xsl:function>
</xsl:stylesheet>

```

## Просмотр результата в браузере:

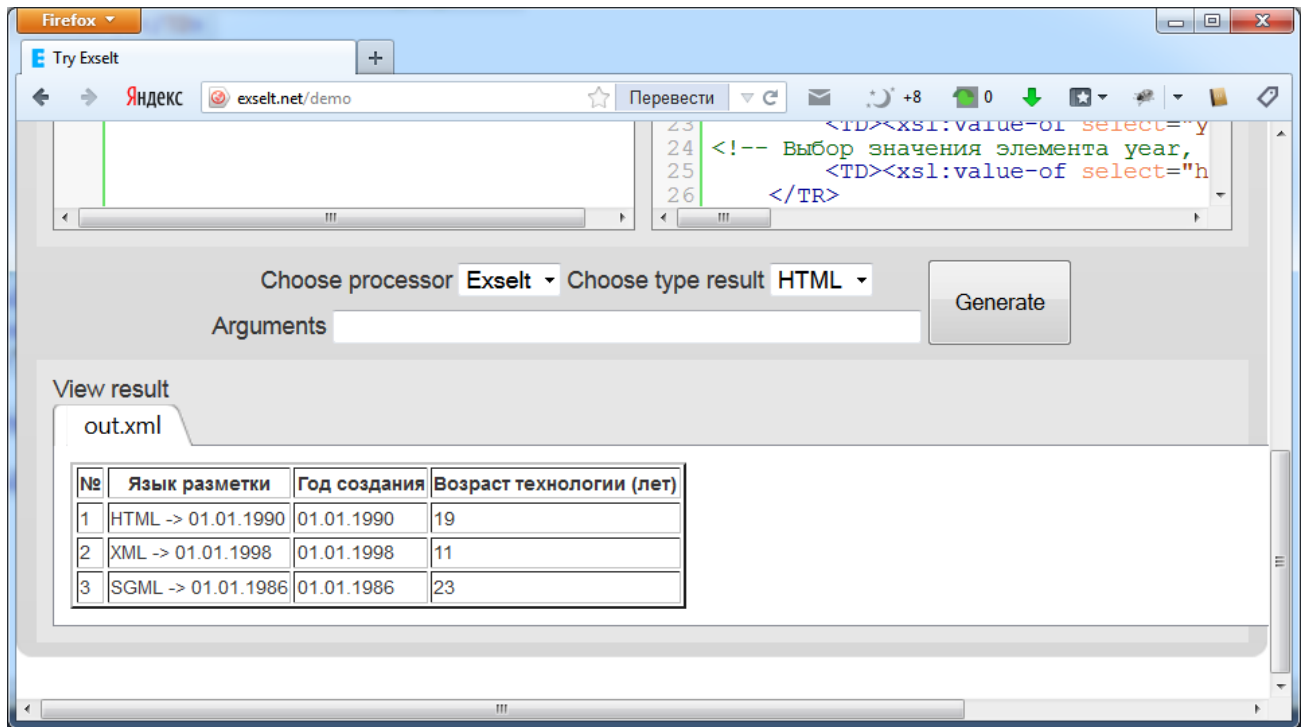


Рис. 2.30. Результат выполнения примера 2.17.

Рассмотрим более подробно описание и вызов функции.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:str="http://example.com/namespace" version="2.0">
```

Открывающий тэг элемента `xsl:stylesheet` содержит объявление пространства имен `xmlns:str="http://example.com/namespace"` которое используется для описания и вызова функции.

```
<!-- Функция вывода в виде ключа и значения -->
<xsl:function name="str:KeyValue" as="xs:string">
  <xsl:param name="key" as="xs:string"/>
  <xsl:param name="value" as="xs:string"/>
  <p>
    <xsl:value-of select="$key" />
    ->
    <xsl:value-of select="$value" />
  </p>
</xsl:function>
```

Функция объявляется с помощью элемента `xsl:function`. Атрибут `name` содержит название функции, а атрибут `as` содержит тип возвращаемого значения (стороковый).

Функция принимает два параметра строкового типа `key` и `value`.

Тело функции представляет собой фрагмент XSLT-преобразования, который выводит ключ и значение в виде «ключ->значение» с использованием директив `xsl:value-of`. Результат работы XSLT-фрагмента, который является телом функции, преобразуется в строку и возвращается из функции в качестве возвращаемого значения.

```
<TD><xsl:value-of select="str:KeyValue(name, year)"/></TD>
```

Выражение `str:KeyValue(name, year)` содержит вызов функции, в качестве аргументов передаются значения элементов `name` и `year` в текущем контексте.

## **2.4 Технология XSLT 3.0**

В настоящее время разрабатывается спецификация XSLT 3.0. В качестве языка запросов в этой версии используется XPath 3.0.

Наиболее существенными дополнениями по сравнению с версией 2.0 являются:

- возможность потоковой обработки документов (streaming mode), в этом случае в память загружается только небольшая часть документа, происходит значительная экономия ресурсов и уменьшение времени обработки (элемент `xsl:stream`);
- возможность параллельной обработки нескольких фрагментов документа (элемент `xsl:fork`);
- возможность динамического формирования и вычисления XPath-запросов (элемент `xsl:evaluate`);
- обработка исключений (элементы `try-catch`).

## **2.5 Материалы для дальнейшего изучения**

Рекомендуется ознакомиться со стандартами XPath 1.0 [XPath, 1999] и XSLT 1.0 [XSLT, 1999] (на русском языке).

Основы технологии XSLT подробно рассмотрены в [Валиков, 2002], [Мангано, 2008].

Также рекомендуется ознакомиться со стандартами XPath 2.0 [XPath, 2007], XSLT 2.0 [XSLT, 2007], XPath 3.0 [XPath, 2014] и XSLT 3.0 [XSLT, 2013].

## **2.6 Контрольные вопросы**

1. Для чего предназначен язык XPath?
2. Какие основные выражения используются в XPath?
3. Чем отличается обращение к элементам от обращения к атрибутам в XPath?
4. Каким образом в XPath можно использовать фильтры и операторы сравнения?
5. Что такое оси выборки в XPath и как их использовать?
6. Что такое контекстный XPath-запрос и чем он отличается от неконтекстного? Почему контекстные XPath-запросы так широко используются?
7. Какие группы функций существуют в XPath?
8. Как использовать арифметические действия в XPath?
9. Как отобразить XML-файл с использованием таблиц стилей CSS? В чем ограничения этого подхода?
10. Для чего предназначена технология XSLT?
11. Какие три варианта преобразований обычно выполняют с помощью XSLT?
12. Что такое XSLT-процессор?
13. Какие элементы технологии XSLT позволяют создавать преобразования, адаптирующиеся к структуре входного документа?

14. Как реализуются циклы и сортировка в XSLT?
15. Как реализуются условия в XSLT?
16. Как реализуется включение стилей в XSLT?
17. Для чего используются переменные в XSLT? В чем их основная особенность?
18. Что такое расширения XSLT? Какие задачи можно решать с их помощью?
19. Как реализуется ключи поиска в XSLT?
20. Как можно реализовать группировку в XSLT 1.0 и 2.0?
21. Какие дополнительные возможности появились в XSLT 2.0?

### 3 Часть 3. Описание структуры документов XML

#### 3.1 Использование DTD для описания структуры документов XML

DTD является одним из способов проверки правильности структуры документа XML. Исторически он появился даже ранее, чем технология XML, так как DTD-описания перешли в XML из SGML.

DTD расшифровывается как Document Type Definition, описание типов документа.

В XML-документах DTD определяет набор используемых элементов, идентифицирует элементы, которые могут использоваться внутри других элементов, определяет возможные атрибуты для каждого элемента.

Подробно DTD-описания рассматриваются в спецификации XML [XML, 2000].

##### 3.1.1 Пример внешнего DTD

Рассмотрим пример использования DTD для документа XML. В этом примере используется внешний DTD, который располагается в отдельном файле. В XML-документ встраивается ссылка на этот внешний файл.

##### Пример 3.1.

##### Файл XML:

```
<?xml version="1.0" encoding="Windows-1251"?>
<!DOCTYPE languages SYSTEM "languages.dtd" >
<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>
<empty attr1="1" attr2="текст"/>
<CDATA_Example>
    <![CDATA[<<<<<<<<<    >>>>>>>>]]>
</CDATA_Example>
</languages>

```

### Файл DTD:

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- В файле DTD могут быть использованы комментарии, если объявлена
инструкция обработки xml -->

<!ELEMENT name (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT howold (#PCDATA)>
<!ELEMENT CDATA_Example (#PCDATA)>

<!ELEMENT language (name, year, howold)>
<!ATTLIST language
    id CDATA #REQUIRED>

<!ELEMENT empty EMPTY>
<!ATTLIST empty
    attr1 CDATA #REQUIRED
    attr2 CDATA #REQUIRED>

<!ELEMENT languages (language+, empty, CDATA_Example)>

```



Проверить соответствие XML-документа файлу DTD можно с использованием XMLPad.

Для этого необходимо открыть документ XML и выбрать пункт меню «XML/Validate».

Если документ «валиден», то есть соответствует DTD, то выдается сообщение об отсутствии ошибок.

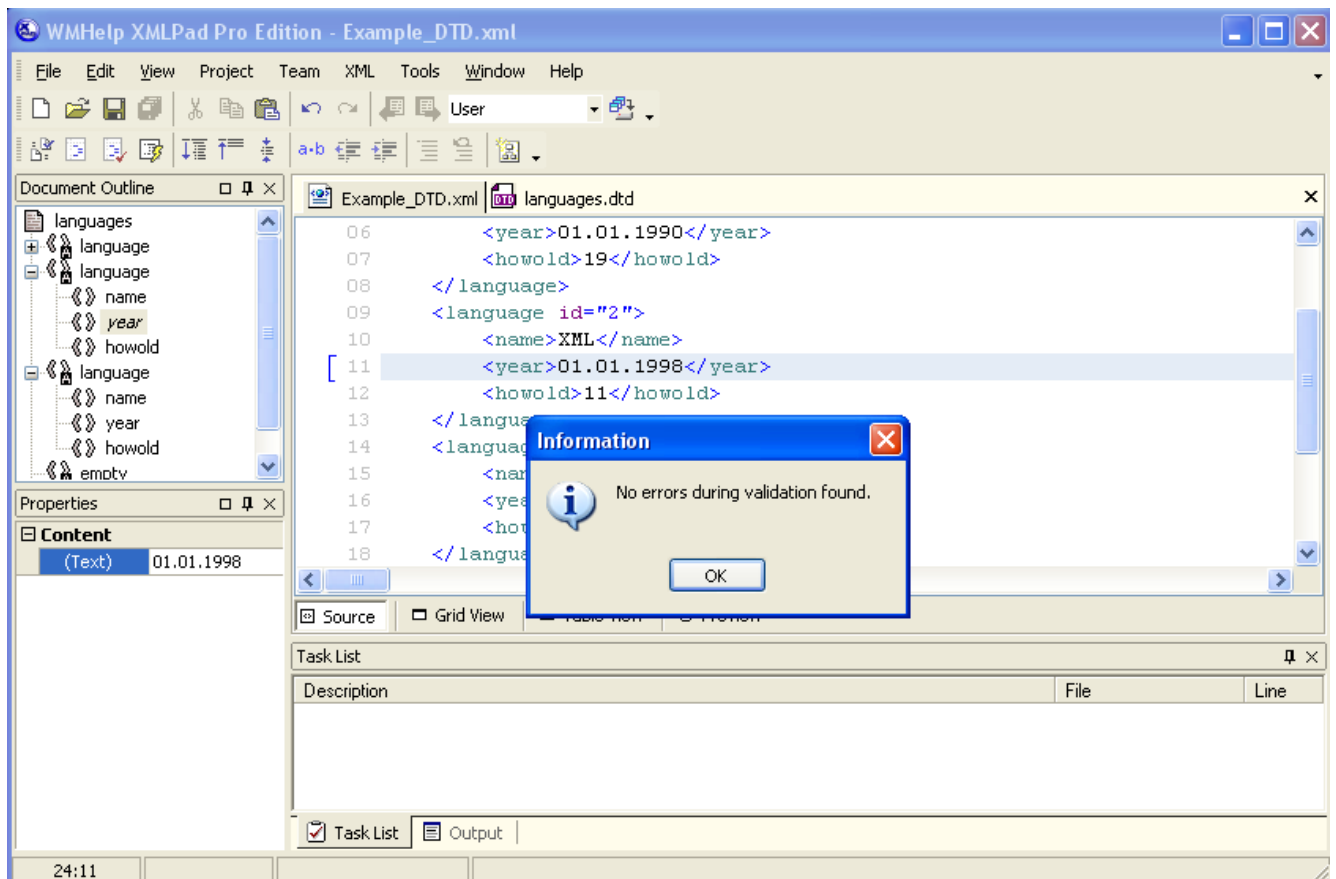


Рис. 3.1. Результат выполнения примера 3.1.

Присоединение DTD-файла к документу XML производится во второй строке XML-документа:

```
<!DOCTYPE languages SYSTEM "languages.dtd" >
```

Имя файла, который содержит DTD – «languages.dtd». К файлу DTD может быть указан полный путь, если он расположен в другом каталоге.

Рассмотрим более подробно текст DTD-описания.

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

`<!-- В файле DTD могут быть использованы комментарии, если объявлена инструкция обработки xml -->`

DTD-документ может быть объявлен как XML-документ, то есть начинаться с инструкции обработки `<?xml . . . ?>`. Это дает возможность использовать в документе XML-комментарии. Если комментарии не нужны, то инструкция обработки `<?xml . . . ?>` может быть пропущена.

```
<!ELEMENT name (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT howold (#PCDATA)>
<!ELEMENT CDATA_Example (#PCDATA)>
```

С помощью команды `!ELEMENT` объявляется элемент в документе. После команды `!ELEMENT` следует название элемента, в скобках указывается содержимое элемента.

`#PCDATA` расшифровывается как «parsed character data», разбираемые символьные данные. Это данные, которые будут разбираться анализатором, например, они могут включать секцию `<![CDATA[ ]]>`.

Элементы, обозначенные как `#PCDATA`, могут включать текст, но не могут включать вложенные элементы.

При объявлении атрибутов также используется обозначение `CDATA` – это данные, которые не будут разбираться анализатором.

```
<!ELEMENT language (name, year, howold)>
```

В элемент `language` могут быть вложены элементы `name`, `year`, `howold`. Они должны следовать именно в таком порядке, каждый элемент встречается один раз.

После названия элемента, а также после выражения в скобках могут встречаться символы «?», «+» и «\*». Эти символы определяют количество вхождений элемента.

«?» – элемент встречается 0 или 1 раз.

«\*» – элемент встречается 0 и более раз (итерация).

«+» – элемент встречается 1 и более раз (позитивная итерация).

То есть используется способ описания, похожий на тот, который применяется в дискретной математике для описания цепочек символов, допускаемых автоматом.

Примеры:

```
<!ELEMENT language (name?, year*, howold+)>
```

```
<!ELEMENT language (name, year, howold)*>
```

Если символ стоит после скобок, то он применяется ко всему выражению в скобках. Например

```
<!ELEMENT language (name?, year*, howold+)+>
```

означает, что элемент `name` должен встречаться 0 или 1 раз, за ним следуют элементы `year`, которые встречаются 0 или более раз, затем следуют элементы `howold`, которые встречаются один или более раз. Последовательность элементов `name?`, `year*`, `howold+` может встречаться один или более раз, это указывает символ «+» после выражения в скобках.

Символ « , » между элементами означает строгое следование элементов друг за другом. Также может использоваться символ « | », который обозначает, что может встречаться один или другой элемент, например:

```
<!ELEMENT language ((name, year, howold) | (year, name, howold))>
```

означает, что в элемент `language` могут быть вложены элементы `name, year, howold` или `year, name, howold`.

```
<!ELEMENT language (((name, year) | (year, name)), howold)>
```

означает, что в элемент `language` могут быть вложены элементы `name, year` или `year, name` и элемент `howold`

```
<!ELEMENT language (name|year|howold)*>
```

означает, что в элемент `language` может быть вложен элемент `name` или `year` или `howold` 0 или более раз. Фактически это означает, что элементы `name, year, howold` могут быть вложены в любой последовательности любое количество раз.

```
<!ATTLIST language
```

```
id CDATA #REQUIRED>
```

С помощью команды `!ATTLIST` определяются атрибуты элемента. После команды `!ATTLIST` следует название элемента, далее перечисляются атрибуты. Для определения атрибута используется три идентификатора.

Первый идентификатор – название атрибута, в нашем случае `id`.

Второй идентификатор определяет тип данных. Чаще всего для обозначения типа используется `CDATA`, то есть любые данные.

Также может использоваться идентификатор `ID`, который определяет уникальное значение атрибута:

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

В качестве типа могут быть перечислены возможные значения атрибута (1 или 2 или 3):

```
<!ATTLIST language id (1|2|3) #REQUIRED>
```

Полный список типов данных приведен в спецификации.

Третий идентификатор определяет обязательность использования атрибута. `#REQUIRED` означает обязательный атрибут, `#IMPLIED` означает необязательный атрибут.

`#FIXED` "значение" определяет, что значение атрибута фиксировано и не может быть изменено. Например

```
<!ATTLIST language id CDATA #FIXED "1">
```

```
<!ELEMENT empty EMPTY>
```

```
<!ATTLIST empty
    attr1 CDATA #REQUIRED
    attr2 CDATA #REQUIRED>
```

Если при объявлении элемента вместо выражения в скобках используется `EMPTY`, то элемент объявляется как пустой, то есть не имеющий содержимого. В этом примере элемент `empty` объявляется как пустой элемент с двумя атрибутами: `attr1` и `attr2`.

```
<!ELEMENT languages (language+, empty, CDATA_Example)>
```

Элемент `languages` содержит один или более элементов `language`, элемент `empty` и элемент `CDATA_Example`.

### 3.1.2 Пример несоответствия документа XML и DTD

Внесем изменения в элемент `<language id="1">` исходного документа. Изменения выделены полужирным шрифтом. В элемент `language` добавлены новый атрибут `attr` и новый элемент `qwerty`.

#### Пример 3.2.

```
<language id="1" attr="123">
    <qwerty>qwerty</qwerty>
    <name>HTML</name>
    <year>01.01.1990</year>
    <howold>19</howold>
</language>
```

При проверке документа на соответствие DTD возникают следующие сообщения об ошибках (в панели Task List):

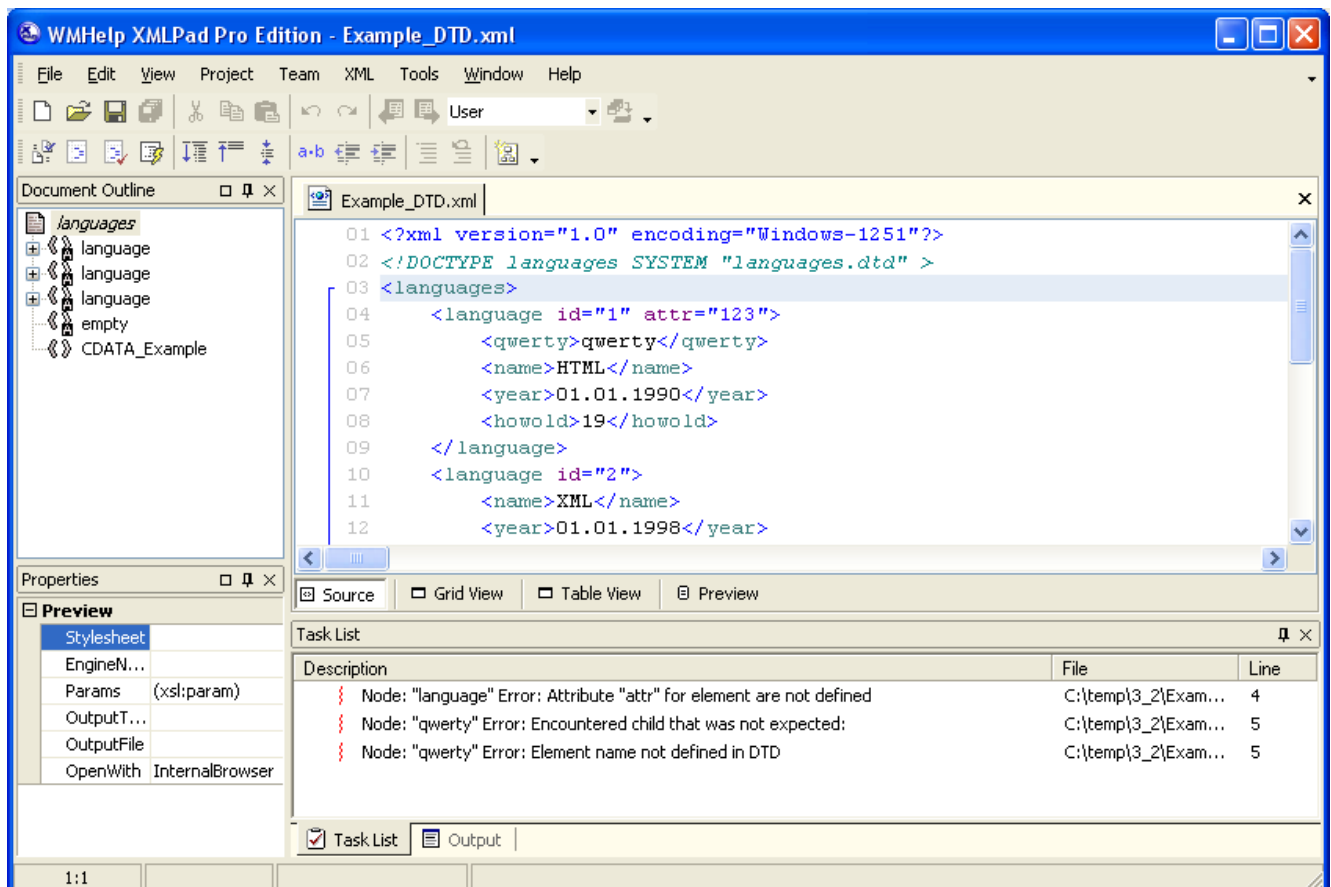


Рис. 3.2. Результат выполнения примера 3.2.

[Оглавление](#)

### 3.1.3 Пример встроенного DTD

DTD может быть встроен в документ XML.

#### Пример 3.3.

```
<?xml version="1.0" encoding="Windows-1251"?>
<!DOCTYPE languages [

<!ELEMENT name (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT howold (#PCDATA)>
<!ELEMENT CDATA_Example (#PCDATA)>

<!ELEMENT language (name, year, howold)>
<!ATTLIST language
    id CDATA #REQUIRED>

<!ELEMENT empty EMPTY>
<!ATTLIST empty
    attr1 CDATA #REQUIRED
    attr2 CDATA #REQUIRED>

<!ELEMENT languages (language+, empty, CDATA_Example)>
]>

<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>
<empty attr1="1" attr2="текст"/>
<CDATA_Example>
    <![CDATA[<<<<<<<<<    >>>>>>>>]]>
</CDATA_Example>
</languages>

```

Результат валидации документа (пункт меню «XML/Validate»):

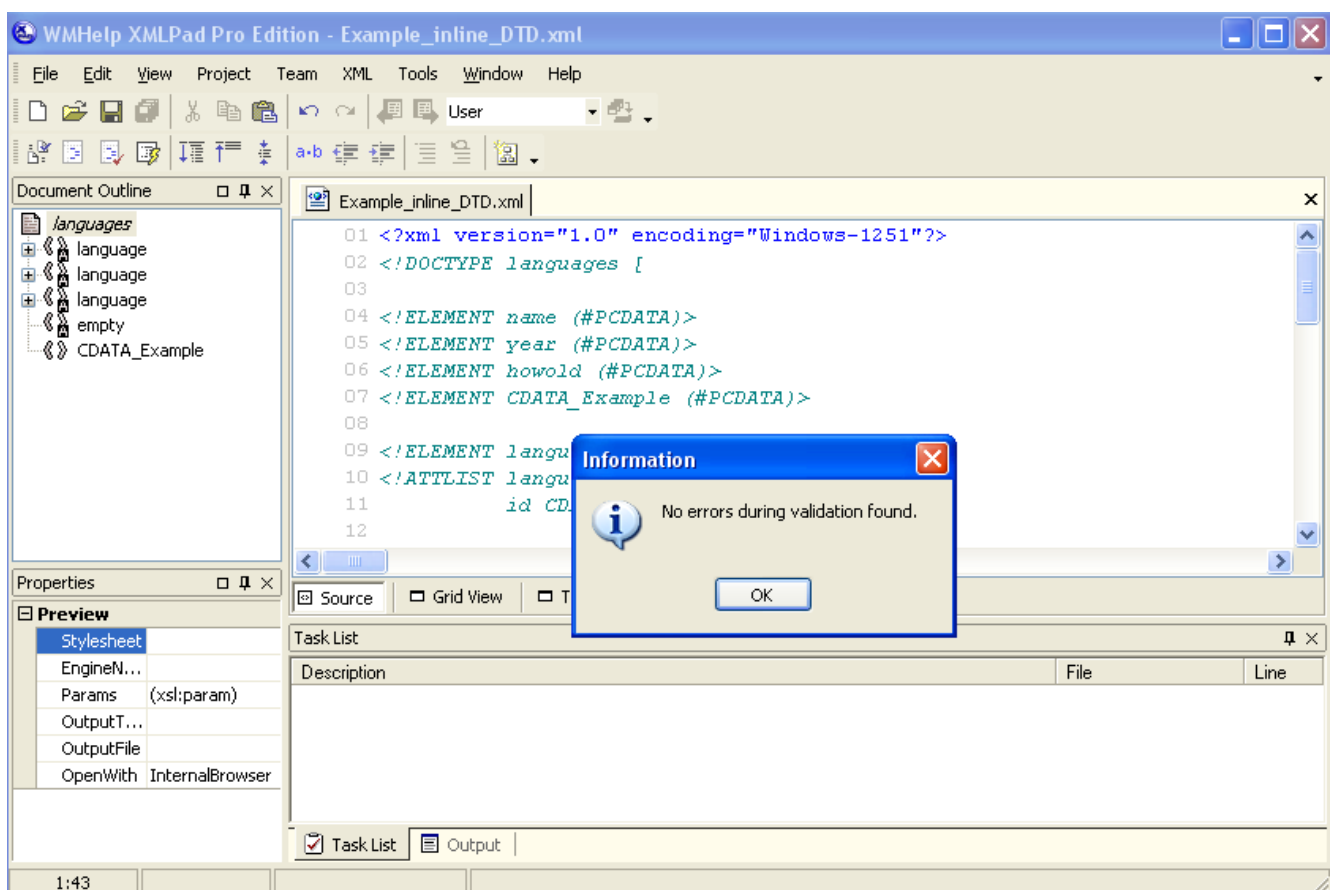


Рис. 3.3. Результат выполнения примера 3.3.

### 3.1.4 Графическое представление DTD

Редактор XMLPad позволяет представить внешнюю DTD в графическом виде. Для этого необходимо открыть файл DTD и выбрать вкладку «Diagram».

Для DTD и XML-схем используется одинаковая графическая нотация. Такая же графическая нотация для DTD и XML-схем используется в редакторе XML SPY.

#### Пример 3.4.

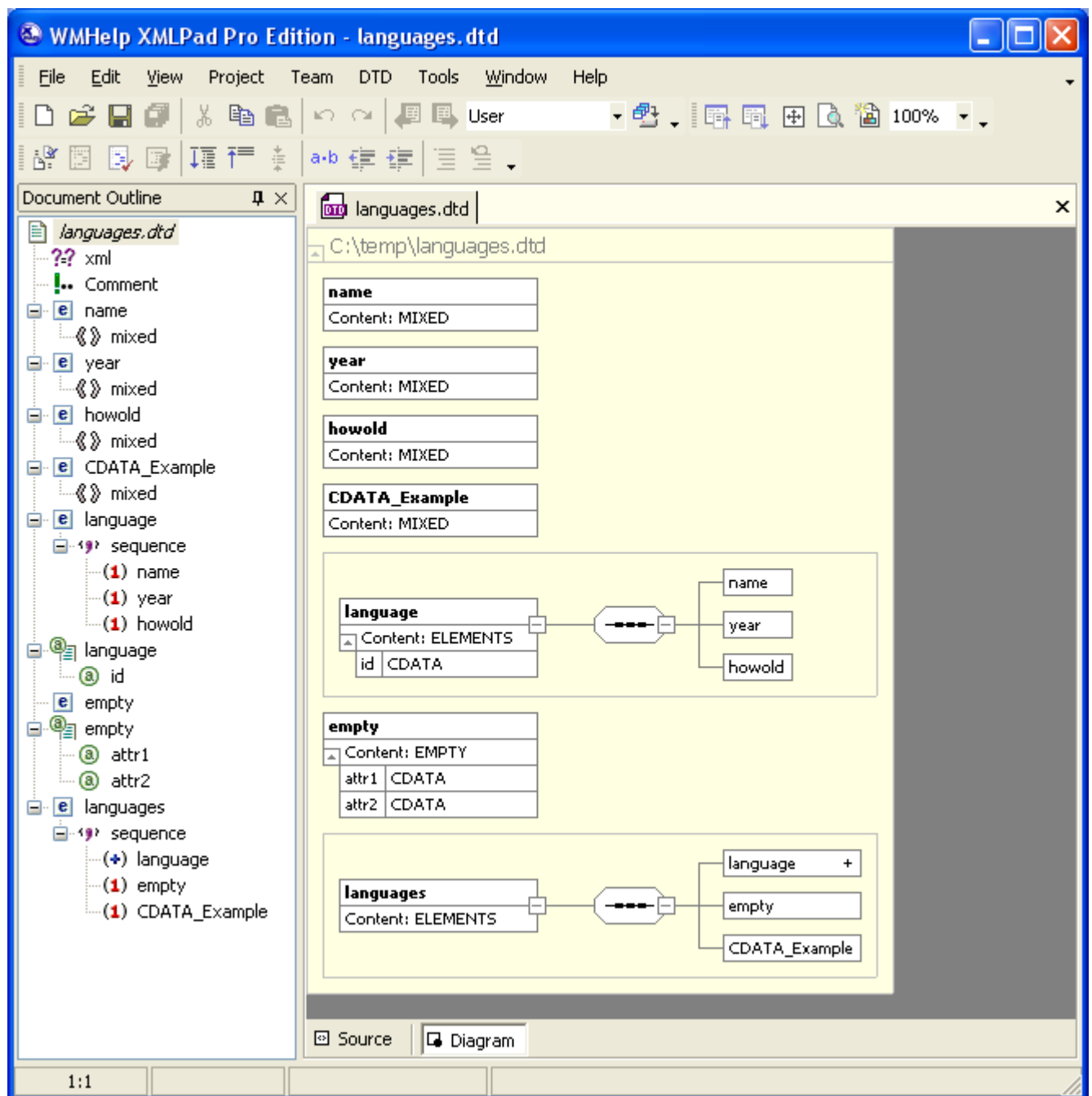


Рис. 3.4. Результат выполнения примера 3.4.



Если объявить элемент language следующим образом:

```
<!ELEMENT language (name? | year+ | howold*)+>
```

то вид диаграммы будет таким:

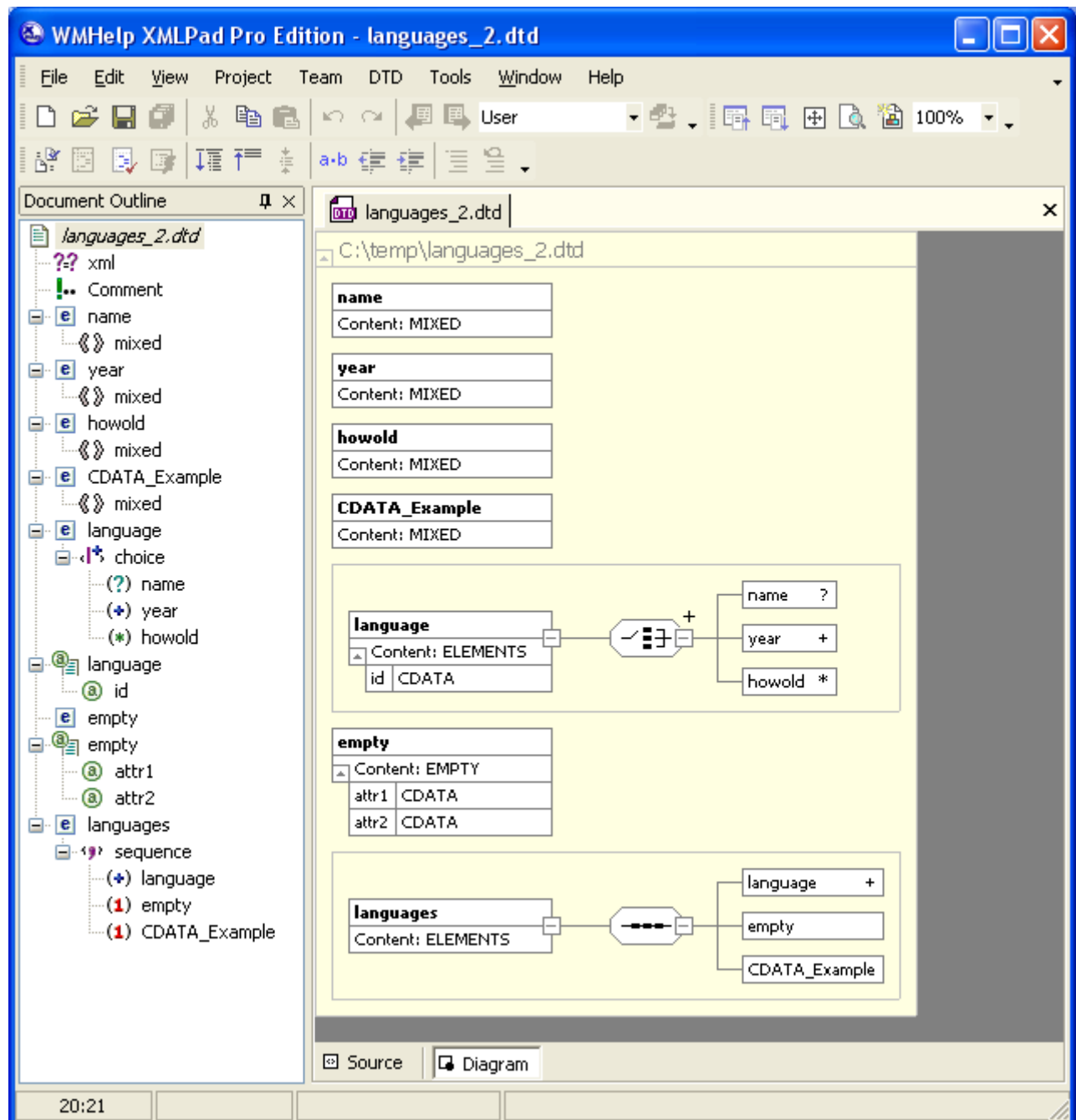


Рис. 3.5. Результат выполнения примера 3.4. Изменение 1.

На диаграмме используются следующие обозначения:



Последовательное соединение элементов, соответствует « , »



Выбор элементов, соответствует « | »

### 3.1.5 Генерация DTD и XML-схемы по XML-документу

XMLPad позволяет сгенерировать DTD или схему XML по документу XML.

Для этого необходимо открыть документ XML, который не связан с DTD или XML-схемой и выбрать пункт меню «XML/Create Schema».

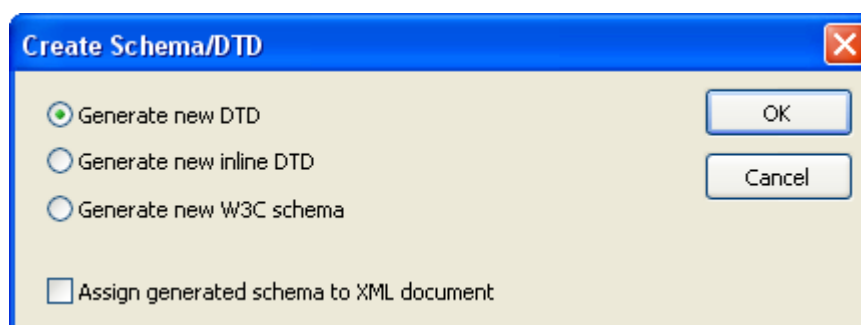


Рис. 3.6. Генерация DTD и XML-схемы по XML-документу.

Существует возможность автоматической генерации DTD, встроенного DTD или XML-схемы.

Также предусмотрена возможность привязки созданного DTD или XML-схемы к XML-документу, при этом в документ XML вставляется соответствующая инструкция.

Необходимо отметить, что полученный DTD или XML-схема почти всегда является «полуфабрикатом», который требуется дорабатывать вручную.

Это происходит по двум причинам. Во-первых, программа генерации может «ошибиться» и сгенерировать не совсем точный результат. Но главная причина в том, что в одном документе, как правило, не присутствуют все возможные варианты, которые должны быть учтены в DTD или XML-схеме.

Например, программа генерации может создать перечисление некоторых вариантов. Но это перечисление не учитывает все возможные случаи, и перечисление необходимо заменить на итерацию.

### 3.1.6 Преобразование DTD в XML-схему и XML-схемы в DTD

В XMLPad существует возможность преобразования DTD в XML-схему (пункт меню «DTD/Convert to XSD»).

В режиме XML-схемы также существует возможность обратного преобразования «XSD/Convert to DTD».

Необходимо учитывать, что программа генерации может сгенерировать результат, требующий корректировки.

В обоих режимах существует возможность генерации примера документа XML, который соответствует DTD или XML-схеме. Пункт меню «DTD/Generate sample XML file» или «XSD/Generate sample XML file».

## 3.2 Использование схем XML для описания структуры документов XML

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

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

### 3.2.1 Пример XML-схемы

#### Пример 3.6.

#### Файл XML:

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

```

<languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Example_schema.xsd">
  <language id="1">
    <name>HTML</name>
    <year>01.01.1990</year>
    <howold>19</howold>
  </language>
  <language id="2">
    <name>XML</name>
    <year>01.01.1998</year>
    <howold>11</howold>
  </language>
  <language id="3">
    <name>SGML</name>
    <year>01.01.1986</year>
    <howold>23</howold>
  </language>
  <empty attr1="1" attr2="текст"/>
  <CDATA_Example>
    <![CDATA[<<<<<<<<<  >>>>>>>>]]>
  </CDATA_Example>
</languages>

```

### Файл XSD:

```

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

  <xsd:complexType name="languageType">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="year" type="xsd:string"/>
      <xsd:element name="howold" type="xsd:integer"/>
    </xsd:sequence>
    <xsd:attribute name="id" use="required" type="xsd:string"/>
  </xsd:complexType>

```

```

<xsd:complexType name="emptyType">
  <xsd:attribute name="attr1" type="xsd:string" use="required"/>
  <xsd:attribute name="attr2" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:element name="languages">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="language" type="languageType"
maxOccurs="unbounded"/>
      <xsd:element name="empty" type="emptyType"/>
      <xsd:element name="CDATA_Example"
type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

</xsd:schema>

```

Проверить соответствие XML-документа схеме, как и в случае DTD, можно с использованием пункта меню «XML/Validate».

Если документ действителен («валиден»), то есть соответствует схеме, то выдается сообщение об отсутствии ошибок. Если документ содержит ошибки, то они выдаются в панели Task List.

Если схема XML набирается вручную в XMLPad, то для проверки правильности синтаксиса можно использовать пункт меню «XSD/Validate».

Также полезным может быть пункт меню «XSD/Change Document Namespace Prefix», который позволяет изменять префикс пространства имен.

Присоединение схемы к документу XML производится при объявлении корневого элемента документа XML.

```

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

```

Для присоединения схемы используются пространства имен. Объявляется пространство имен со значением "http://www.w3.org/2001/XMLSchema-instance", это пространство имен используется для подключения в документ схемы XML. Префикс пространства имен может выбираться произвольно, в нашем примере для префикса выбрано значение xsi.

Далее указывается атрибут xsi:noNamespaceSchemaLocation. Этот атрибут принадлежит указанному пространству имен, на что указывает префикс xsi перед названием атрибута. То есть этот атрибут является служебным, и относится не к документу, а используется для подключения XML-схемы. Значением атрибута является URI файла, содержащего схему XML.

Это наиболее простой способ подключения схемы, далее будут рассмотрены другие способы.

Рассмотрим более подробно текст XML-схемы.

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

XML-схема является XML-документом, корневым элементом является элемент schema.

Элементы схемы принадлежат пространству имен "http://www.w3.org/2001/XMLSchema". В качестве префикса пространства имен обычно используется xsd или xs.

```
<xsd:complexType name="languageType">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="year" type="xsd:string"/>
    <xsd:element name="howold" type="xsd:integer"/>
  </xsd:sequence>
  <xsd:attribute name="id" use="required" type="xsd:string"/>
</xsd:complexType>
```

Объявление сложного (составного) типа с названием languageType. Этот тип определяет последовательность элементов (xsd:sequence). В последовательность

входят три элемента name, year, howold, они простого строкового (xsd:string) или целого (xsd:integer) типа.

Также определяется атрибут id, обязательный (use="required"), строкового типа (type="xsd:string").

Составной тип определяет содержимое какого-либо элемента, но название самого элемента в типе не указывается. Несколько элементов с разными названиями могут быть одного типа.

```
<xsd:complexType name="emptyType">
  <xsd:attribute name="attr1" type="xsd:string" use="required"/>
  <xsd:attribute name="attr2" type="xsd:string" use="required"/>
</xsd:complexType>
```

Объявление типа с названием emptyType. Этот тип соответствует пустому элементу, так как в нем не объявлено содержимое элемента. В этом типе объявлено два атрибута.

Типы, объявленные на уровне схемы, называются глобальными типами.

```
<xsd:element name="languages">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="language" type="languageType"
maxOccurs="unbounded"/>
      <xsd:element name="empty" type="emptyType"/>
      <xsd:element name="CDATA_Example"
type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

В схеме также объявлен глобальный элемент languages. Обычно в качестве глобального объявляют корневой элемент документа.

В `xsd:element` вложен `xsd:complexType`. Это означает, что элемент `languages` составного типа. Поскольку на вложенный тип нет ссылок из других элементов, то атрибут `name` для `xsd:complexType` не используется, то есть тип является анонимным.

Составной тип содержит последовательность из трех элементов.

Элемент `language` типа `languageType` может встречаться неограниченное количество раз (`maxOccurs="unbounded"`). Атрибут `maxOccurs` указывает максимальное количество вхождений элемента, атрибут `minOccurs` указывает минимальное количество вхождений элемента. Атрибут может содержать число вхождений ("0", "1" и т.д.) или значение "unbounded" (неограниченное количество вхождений).

В последовательность также входят элемент `empty` типа `emptyType` и элемент `CDATA_Example` типа `xsd:string`. Каждый из этих элементов должен встречаться один раз.

### 3.2.2 Графическое представление схемы XML

Редактор XMLPad позволяет представить XML-схему в графическом виде. Для этого необходимо открыть файл со схемой и выбрать вкладку «Diagram». На диаграмме показаны глобальные типы и глобальный элемент.



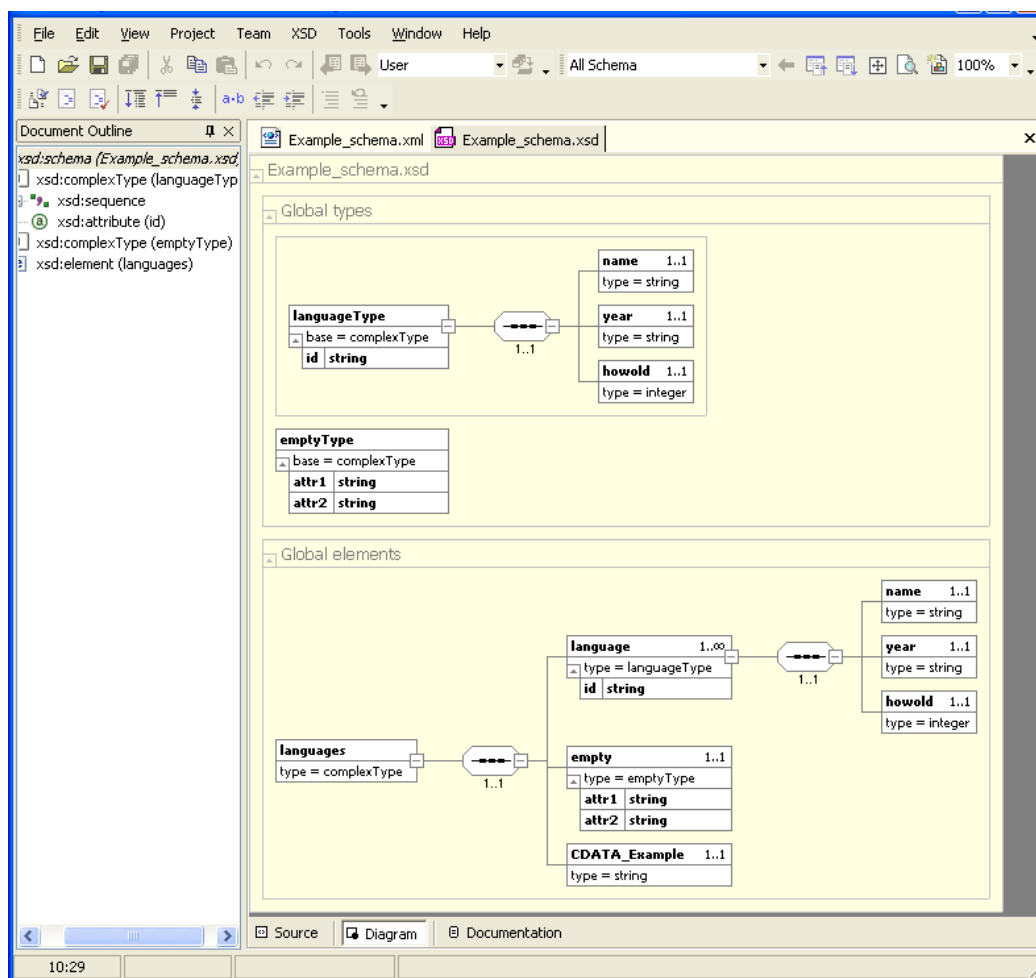


Рис. 3.7. Графическое представление схемы XML из примера 3.6.

Особенностью графического представления схемы является то, что некоторые фрагменты могут повторяться. Это связано с использованием типов. Например, элемент `language` типа `languageType`, поэтому содержимое составного типа `languageType` и элемента `language` одинаково.

Для генерации HTML-документа, содержащего подробное описание схемы необходимо выбрать вкладку «Documentation». При этом в текущем каталоге будет создан HTML-документ, содержащий документацию.

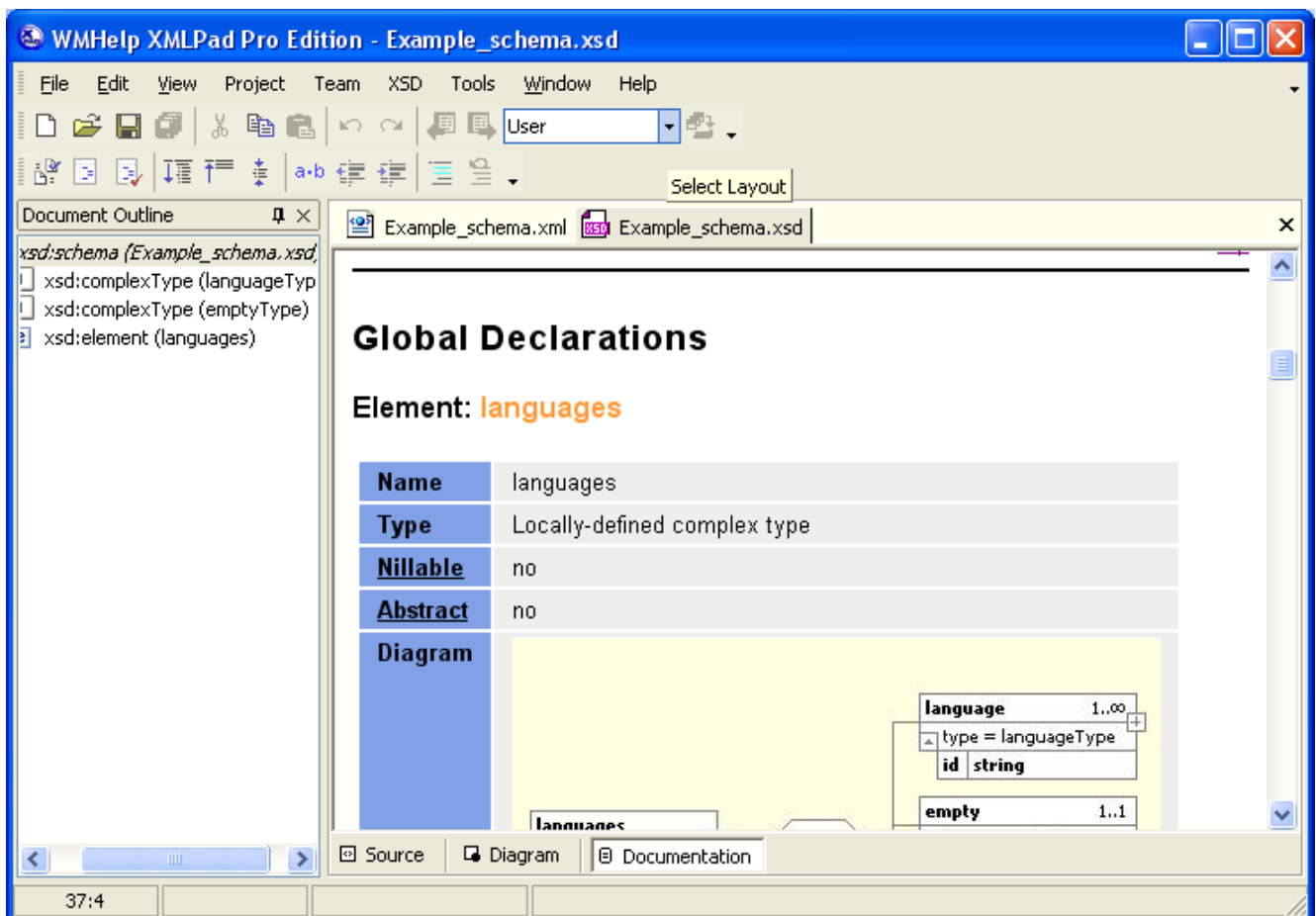


Рис. 3.8. Графическое представление схемы XML из примера 3.6. Формирование документации.

### 3.2.3 Использование простых типов и ограничений

Простыми (simple type) называются типы, которые определяют значение простого элемента или атрибута. Простой элемент содержит атомарное значение и не может иметь вложенных элементов.

Тип данных, который определяет элемент, в который вложены другие элементы, называется сложным или составным типом данных (complex type).

На следующем рисунке показана иерархия типов данных, приведенная в спецификации.

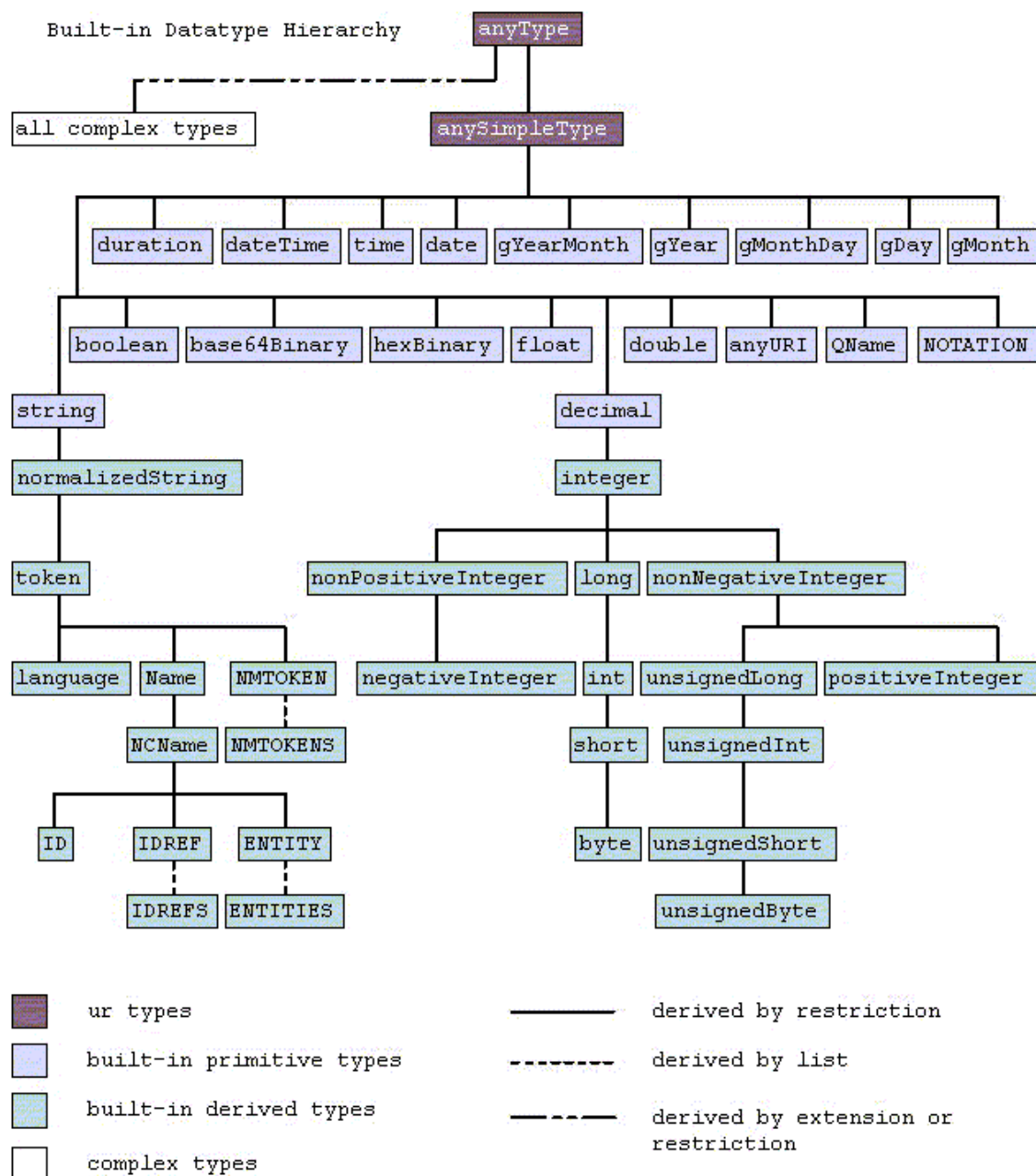


Рис. 3.9. Иерархия типов данных XML-схем.

Спецификация XML-схем позволяет вводить производные простые типы, накладывая ограничения на базовые простые типы.

Эти ограничения также называют граниями, фасетами (facets).

Таблица 3.1. Ограничения строковых типов данных.

length	Ограничение на длину строки
minLength	Ограничение на длину строки снизу
maxLength	Ограничение на длину строки сверху
pattern	Задание шаблона строки регулярным выражением
enumeration	Перечисление элементов

Таблица 3.2. Ограничения числовых типов данных.

maxInclusive	Больше или равно
maxExclusive	Больше
minExclusive	Меньше
minInclusive	Меньше или равно
totalDigits	Количество цифр в числе
fractionDigits	Количество цифр в дробной части

Пример использования ограничений.

### Пример 3.7.

#### Файл XML:

```
<?xml version="1.0" encoding="Windows-1251"?>
<!-- Примеры ограничений для простых типов -->

<Root_Element xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="facets.xsd">

    <Example_Length>1234</Example_Length>
    <Example_Length_MinMax>12345</Example_Length_MinMax>
    <Example_Length_MinMax>1234567</Example_Length_MinMax>

    <Example_Pattern>aaaaabbbbbccccbbbbbbaaaaaa</Example_Pattern>

    <domain>gov</domain>
```

[Оглавление](#)

```
<domain>edu</domain>
```

```
<Example_Inclusive>1</Example_Inclusive>
```

```
<Example_Inclusive>5</Example_Inclusive>
```

```
<Example_Inclusive>10</Example_Inclusive>
```

```
<Example_Exclusive>2</Example_Exclusive>
```

```
<Example_Exclusive>9</Example_Exclusive>
```

```
<Example_Digits>123.45</Example_Digits>
```

```
</Root_Element>
```

### Файл XSD:

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

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

```
  <xsd:simpleType name="Example_Length_Type">
```

```
    <xsd:restriction base="xsd:string">
```

```
      <xsd:length value='4' />
```

```
    </xsd:restriction>
```

```
  </xsd:simpleType>
```

Простой тип с названием Example\_Length\_Type создается как ограничение (xsd:restriction), накладываемое на базовый строковый тип (xsd:string). В элемент xsd:restriction вкладываются элементы ограничений, в данном случае ограничение на общую длину строки.

```
<xsd:simpleType name="Example_Length_MinMax_Type">
```

```
  <xsd:restriction base="xsd:string">
```

```
    <xsd:minLength value='5' />
```

```
    <xsd:maxLength value='7' />
```

```
  </xsd:restriction>
```

```
</xsd:simpleType>
```

```

<xsd:simpleType name="Example_Pattern_Type">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="(a|b|c) *"/>
  </xsd:restriction>
</xsd:simpleType>

```

```

<xsd:simpleType name="Example_enumeration_Type">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value='gov' />
    <xsd:enumeration value='edu' />
    <xsd:enumeration value='com' />
  </xsd:restriction>
</xsd:simpleType>

```

```

<xsd:simpleType name="Example_Inclusive_Type">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value='1' />
    <xsd:maxInclusive value='10' />
  </xsd:restriction>
</xsd:simpleType>

```

```

<xsd:simpleType name="Example_Exclusive_Type">
  <xsd:restriction base="xsd:integer">
    <xsd:minExclusive value='1' />
    <xsd:maxExclusive value='10' />
  </xsd:restriction>
</xsd:simpleType>

```

```

<xsd:simpleType name="Example_Digits_Type">
  <xsd:restriction base="xsd:decimal">
    <xsd:totalDigits value='5' />
    <xsd:fractionDigits value='2' />
  </xsd:restriction>
</xsd:simpleType>

```

```

<xsd:complexType name="Root_Element_Type">
  <xsd:sequence>
    <xsd:element name="Example_Length"
type="Example_Length_Type" maxOccurs="unbounded"/>
    <xsd:element name="Example_Length_MinMax"
type="Example_Length_MinMax_Type" maxOccurs="unbounded"/>
    <xsd:element name="Example_Pattern"
type="Example_Pattern_Type"/>
    <xsd:element name="domain"
type="Example_enumeration_Type" maxOccurs="unbounded"/>

    <xsd:element name="Example_Inclusive"
type="Example_Inclusive_Type" maxOccurs="unbounded"/>
    <xsd:element name="Example_Exclusive"
type="Example_Exclusive_Type" maxOccurs="unbounded"/>
    <xsd:element name="Example_Digits"
type="Example_Digits_Type" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

```

Составной тип, в котором определяются элементы для тестирования ограничений.

```

<xsd:element name="Root_Element" type="Root_Element_Type"/>
</xsd:schema>

```

Определение корневого элемента документа.

### 3.2.4 Списки и объединения

Под списком в XML понимается возможность хранения списка значений в одном элементе XML. Значения в списке разделяются пробелами. Пример списка:

```
<Example_List>1 2 3 4 5 11 12 13 14 15</Example_List>
```

Объединение позволяет хранить в элементе значение одного или другого типа. Пример объединения, позволяющего хранить в элементе действительное число или список целых:

```
<Example_Union>1 2 3 4 5</Example_Union>
<Example_Union>123.123</Example_Union>
```

Списки и объединения относятся к простым типам. К спискам и объединениям можно применять ограничения.

### Пример 3.8.

#### Файл XML:

```
<?xml version="1.0" encoding="Windows-1251"?>
<!-- Примеры списков и объединений -->

<Root_Element xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="list_union.xsd">

  <Example_List_Integer>1 2 3 4 5 11 12 13 14
15</Example_List_Integer>

  <Example_List_Integer_Length>1 2 3</Example_List_Integer_Length>

  <Example_List_String>строка1 строка2
строка3</Example_List_String>

  <Example_List_String_Enum>AAA BBB CCC</Example_List_String_Enum>

  <Example_Union>1 2 3 4 5</Example_Union>
  <Example_Union>123</Example_Union>

  <Example_Union2>333</Example_Union2>
  <Example_Union2>3.333</Example_Union2>

</Root_Element>
```

#### Файл XSD:



```
<?xml version="1.0" encoding="Windows-1251"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
  <xsd:simpleType name="Example_List_Integer_Type">
    <xsd:list itemType='xsd:integer' />
  </xsd:simpleType>
```

Элемент `xsd:list` определяет список, в атрибуте `itemType` указывается тип элемента списка.

```
  <xsd:simpleType name="Example_List_Integer_Length_Type">
    <xsd:restriction base="Example_List_Integer_Type">
      <xsd:length value='5' />
    </xsd:restriction>
  </xsd:simpleType>
```

Применяется ограничение на количество элементов в списке.

```
  <xsd:simpleType name="Example_List_String_Type">
    <xsd:list itemType='xsd:string' />
  </xsd:simpleType>
```

```
  <xsd:simpleType name="Example_List_String_Enum_Type">
    <xsd:restriction base="Example_List_String_Type">
      <xsd:enumeration value='AAA BBB CCC' />
      <xsd:enumeration value='BBB' />
      <xsd:enumeration value='CCC' />
    </xsd:restriction>
  </xsd:simpleType>
```

```
  <xsd:simpleType name="Example_Union1_Type">
    <xsd:union memberTypes="xsd:integer
Example_List_Integer_Type" />
  </xsd:simpleType>
```

Элемент `xsd:union` определяет объединение. В атрибуте `memberTypes` указывается список типов, которые могут входить в объединение. В этом примере в объединение могут входить целое число или список целых.

```
<xsd:simpleType name="Example_Union2_Type">
  <xsd:union memberTypes="xsd:integer xsd:double"/>
</xsd:simpleType>
```

Элемент может быть целым числом или действительным числом.

```
<xsd:complexType name="Root_Element_Type">
  <xsd:sequence>
    <xsd:element name="Example_List_Integer"
type="Example_List_Integer_Type"/>
    <xsd:element name="Example_List_Integer_Length"
type="Example_List_Integer_Length_Type"/>
    <xsd:element name="Example_List_String"
type="Example_List_String_Type"/>
    <xsd:element name="Example_List_String_Enum"
type="Example_List_String_Enum_Type"/>
    <xsd:element name="Example_Union"
type="Example_Union1_Type" maxOccurs="unbounded"/>
    <xsd:element name="Example_Union2"
type="Example_Union2_Type" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

Составной тип, в котором определяются элементы для создания списков и объединений.

```
<xsd:element name="Root_Element" type="Root_Element_Type"/>
</xsd:schema>
```

Определение корневого элемента документа.

### 3.2.5 Простые элементы с атрибутами

В следующем примере создаются элемент с простым содержимым, содержащий атрибуты, и пустой элемент, содержащий атрибуты. Также показано использование типа `xsd:anyType`, который позволяет объявлять элементы произвольного типа с произвольным содержимым.

#### Пример 3.9.

##### Файл XML:

```
<?xml version="1.0" encoding="Windows-1251"?>
<!-- Примеры сложных типов -->

<Root_Element xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="compl_attrib.xsd">

    <Example_IntegerElement attr1="333"
attr2="string">333</Example_IntegerElement>

    <Example_EmptyElement attr1="333" attr2="string"/>

    <Example_anyType1 NotDefinedAttribute="NotDefinedAttribute">123
текст</Example_anyType1>

    <Example_anyType2 NotDefinedAttribute="NotDefinedAttribute">123
текст <child>123</child></Example_anyType2>

</Root_Element>
```

##### Файл XSD:

```
<?xml version="1.0" encoding="Windows-1251"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:complexType name="Example_IntegerElement_Type">
        <xsd:simpleContent>
            <xsd:extension base="xsd:integer">
                <xsd:attribute name="attr1" type="xsd:integer"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:schema>
```

```

        <xsd:attribute name="attr2" type="xsd:string"/>
    </xsd:extension>
</xsd:simpleContent>
</xsd:complexType>

```

Объявление типа элемента с двумя атрибутами, который содержит целое число. Этот тип объявляется как составной (xsd:complexType), но у него простое содержимое (xsd:simpleContent). Содержимое элемента базируется на целом числе (xsd:extension base="xsd:integer"). Элемент содержит описание атрибутов.

```

<xsd:complexType name="Example_EmptyElement_Type">
    <xsd:attribute name="attr1" type="xsd:integer"/>
    <xsd:attribute name="attr2" type="xsd:string"/>
</xsd:complexType>

```

Объявление типа, соответствующего пустому элементу с двумя атрибутами.

```

<xsd:complexType name="Root_Element_Type">
    <xsd:sequence>
        <xsd:element name="Example_IntegerElement"
type="Example_IntegerElement_Type"/>
        <xsd:element name="Example_EmptyElement"
type="Example_EmptyElement_Type"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="Example_anyType1" type="xsd:anyType"/>
<xsd:element name="Example_anyType2"/>

```

Использование типа xsd:anyType позволяет объявлять элементы произвольного типа с произвольным содержимым. Если тип не указан, то по умолчанию это также xsd:anyType.

```

    </xsd:sequence>
</xsd:complexType>
<xsd:element name="Root_Element" type="Root_Element_Type"/>
</xsd:schema>

```

### 3.2.6 Использование сложных (составных) типов

Сложные (составные) типы формируются с помощью следующих конструкций:

- `xsd:sequence` – определяет последовательность вложенных элементов. Соответствует символу « , » в DTD.
- `xsd:choice` – определяет выбор элементов. Соответствует символу « | » в DTD.
- `xsd:all` – определяет следование всех элементов в любом порядке. Каждый элемент может содержаться ноль или один раз. `xsd:all` используется редко.

Эти конструкции формируют «модель содержимого» элемента XML-документа.

Элементы `xsd:sequence` и `xsd:choice` могут быть вложены друг в друга, что соответствует комбинации « , » и « | » в DTD.

Аналогом символов «?», «\*» и «+» из DTD в схемах XML являются атрибуты `minOccurs` и `maxOccurs`, которые определяют минимальное и максимальное количество вхождений элемента.

Каждый из атрибутов может содержать непосредственное число вхождений ("0", "1" и т.д.) или значение "unbounded" (неограниченное количество вхождений).

#### Пример 3.10.

##### Файл XML:

```
<?xml version="1.0" encoding="Windows-1251"?>
<!-- Языки разметки -->
<languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="compl_content_scl.xsd">

    <language1>
        <name>HTML</name>
        <name1>HTML</name1>
```

```

</language1>

<language1>
  <year>01.01.1990</year>
  <howold1>14</howold1>
</language1>
<!-- ++++++ -->
<language2>
  <name>SGML</name>
  <year>01.01.1986</year>
  <howold>18</howold>
</language2>

<language2>
  <name1>SGML</name1>
  <year1>01.01.1986</year1>
  <howold1>18</howold1>
</language2>
<!-- ++++++ -->
<language3>
  <NAME>SGML</NAME>
  <year>01.01.1986</year>
  <NAME1>SGML</NAME1>
  <year1>01.01.1986</year1>
</language3>

<language3>
  <name>SGML</name>
  <howold>18</howold>
  <name1>SGML</name1>
  <howold1>18</howold1>
</language3>
</languages>

```

### Файл XSD:

```

<?xml version="1.0" encoding="Windows-1251"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <!-- Последовательность выборов -->
    <xsd:complexType name="languageType_1">
        <xsd:sequence>

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

            <xsd:choice>
                <xsd:element name="name1" type="xsd:string"/>
                <xsd:element name="year1" type="xsd:string"/>
                <xsd:element name="howold1" type="xsd:integer"/>
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType>

    <!-- Выбор последовательностей -->
    <xsd:complexType name="languageType_2">
        <xsd:choice>

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

            <xsd:sequence>
                <xsd:element name="name1" type="xsd:string"/>
                <xsd:element name="year1" type="xsd:string"/>
                <xsd:element name="howold1" type="xsd:integer"/>
            </xsd:sequence>
        </xsd:choice>
    </xsd:complexType>

```

```

        </xsd:sequence>

    </xsd:choice>
</xsd:complexType>

<!-- Последовательность выборов последовательностей -->
<!-- Возможно определить также выбор последовательностей выборов -->
<xsd:complexType name="languageType_3">
    <xsd:sequence>

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

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

        <xsd:choice>
            <xsd:sequence>
                <xsd:element name="NAME1" type="xsd:string"/>
                <xsd:element name="year1" type="xsd:string"/>
            </xsd:sequence>

            <xsd:sequence>
                <xsd:element name="name1" type="xsd:string"/>
                <xsd:element name="howold1" type="xsd:integer"/>
            </xsd:sequence>
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>

```



```

<xsd:element name="languages">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="language1"
type="languageType_1" maxOccurs="unbounded"/>
      <xsd:element name="language2"
type="languageType_2" maxOccurs="unbounded"/>
      <xsd:element name="language3"
type="languageType_3" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>

```

Более сложный пример, в котором используются атрибуты `minOccurs` и `maxOccurs`.

### Пример 3.11.

#### Файл XML:

```

<?xml version="1.0" encoding="Windows-1251"?>
<!-- Языки разметки -->
<languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="compl_content_sc2.xsd">

```

```

  <language_1_1>
    <name>HTML</name>
    <year>01.01.1986</year>
    <howold>18</howold>

    <name>HTML</name>
    <year>01.01.1986</year>
    <howold>18</howold>
  </language_1_1>

  <language_1_2>

```

```

<name>HTML</name>
<name>HTML</name>
<year>01.01.1986</year>
<year>01.01.1986</year>
<howold>10</howold>
<howold>10</howold>
</language_1_2>

```

```

<language_1_3>
  <name>HTML</name>
  <name>HTML</name>
  <year>01.01.1986</year>
  <year>01.01.1986</year>
  <howold>10</howold>
  <howold>10</howold>

  <name>HTML</name>
  <name>HTML</name>
</language_1_3>

```

```

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

```

```

<language_2_1>
  <name1>HTML</name1>
  <name>HTML</name>
  <year>01.01.1986</year>
  <year>01.01.1986</year>
  <howold>10</howold>
  <howold>10</howold>

  <name>HTML</name>
  <year>01.01.1986</year>
  <howold>10</howold>

  <year>01.01.1986</year>

```

```

    <howold>10</howold>
    <name>HTML</name>
</language_2_1>

```

```

<language_2_2>
    <name>HTML</name>
    <name>HTML</name>
</language_2_2>

```

```

<language_2_3>
    <name>HTML</name>
    <name>HTML</name>
    <year>01.01.1986</year>
    <year>01.01.1986</year>
    <howold>10</howold>
    <howold>10</howold>

    <name>HTML</name>
    <year>01.01.1986</year>
    <howold>10</howold>

    <year>01.01.1986</year>
    <howold>10</howold>
    <name>HTML</name>
</language_2_3>

```

```

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

```

```

<language_3_1>
    <name1>HTML</name1>
    <name>HTML</name>
    <year>01.01.1986</year>
    <year>01.01.1986</year>
    <howold>10</howold>
    <howold>10</howold>

```

```

    <name>HTML</name>
    <year>01.01.1986</year>
    <howold>10</howold>

    <year>01.01.1986</year>
    <howold>10</howold>
    <name>HTML</name>
</language_3_1>

<language_3_2>
    <name1>HTML</name1>
    <name>HTML</name>
    <year>01.01.1986</year>
    <year>01.01.1986</year>
    <howold>10</howold>
    <howold>10</howold>

    <name>HTML</name>
    <year>01.01.1986</year>
    <howold>10</howold>

    <year>01.01.1986</year>
    <howold>10</howold>
    <name>HTML</name>
</language_3_2>
</languages>

```

### Файл XSD:

```

<?xml version="1.0" encoding="Windows-1251"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <!-- Может повторяться вся последовательность -->
    <xsd:complexType name="languageType_1_1">
        <xsd:sequence minOccurs="1" maxOccurs="unbounded">

```

```

        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="year" type="xsd:string"/>
        <xsd:element name="howold" type="xsd:integer"/>
    </xsd:sequence>
</xsd:complexType>

<!-- Может повторяться элемент в последовательности -->
<xsd:complexType name="languageType_1_2">
    <xsd:sequence>
        <xsd:element name="name" type="xsd:string"
minOccurs="1" maxOccurs="unbounded"/>
        <xsd:element name="year" type="xsd:string"
minOccurs="1" maxOccurs="unbounded"/>
        <xsd:element name="howold" type="xsd:integer"
minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<!-- Может повторяться вся последовательность и элемент в
последовательности -->
<xsd:complexType name="languageType_1_3">
    <xsd:sequence minOccurs="1" maxOccurs="unbounded">
        <xsd:element name="name" type="xsd:string"
minOccurs="1" maxOccurs="unbounded"/>
        <xsd:element name="year" type="xsd:string"
minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="howold" type="xsd:integer"
minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<!-- ++++++ -->
<!-- Любой элемент может повторяться произвольное количество раз -->
<xsd:complexType name="languageType_2_1">
    <xsd:choice minOccurs="1" maxOccurs="unbounded">
        <xsd:element name="name" type="xsd:string"/>

```

```

        <xsd:element name="year" type="xsd:string"/>
        <xsd:element name="howold" type="xsd:integer"/>
    </xsd:choice>
</xsd:complexType>

```

<!-- Выбор одного элемента из группы, элемент может повторяться произвольное количество раз -->

```

    <xsd:complexType name="languageType_2_2">
        <xsd:choice>
            <xsd:element name="name" type="xsd:string"
minOccurs="1" maxOccurs="unbounded"/>
            <xsd:element name="year" type="xsd:string"
minOccurs="1" maxOccurs="unbounded"/>
            <xsd:element name="howold" type="xsd:integer"
minOccurs="1" maxOccurs="unbounded"/>
        </xsd:choice>
    </xsd:complexType>

```

<!-- Любой элемент может повторяться произвольное количество раз -->

```

    <xsd:complexType name="languageType_2_3">
        <xsd:choice minOccurs="1" maxOccurs="unbounded">
            <xsd:element name="name" type="xsd:string"
minOccurs="1" maxOccurs="unbounded"/>
            <xsd:element name="year" type="xsd:string"
minOccurs="1" maxOccurs="unbounded"/>
            <xsd:element name="howold" type="xsd:integer"
minOccurs="1" maxOccurs="unbounded"/>
        </xsd:choice>
    </xsd:complexType>

```

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

<!-- Избыточное описание -->

```

<xsd:complexType name="languageType_3_1">
    <xsd:sequence minOccurs="1" maxOccurs="unbounded">
        <xsd:choice>
            <xsd:element name="name" type="xsd:string"/>

```

```

    <xsd:element name="year" type="xsd:string"/>
    <xsd:element name="howold" type="xsd:integer"/>
  </xsd:choice>
</xsd:sequence>
</xsd:complexType>

```

```

<xsd:complexType name="languageType_3_2">
  <xsd:sequence>
    <xsd:choice minOccurs="1" maxOccurs="unbounded">
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="year" type="xsd:string"/>
      <xsd:element name="howold" type="xsd:integer"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>

```

```

<xsd:element name="languages">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="language_1_1" type="languageType_1_1"
maxOccurs="unbounded"/>
      <xsd:element name="language_1_2" type="languageType_1_2"
maxOccurs="unbounded"/>
      <xsd:element name="language_1_3" type="languageType_1_3"
maxOccurs="unbounded"/>
      <xsd:element name="language_2_1" type="languageType_2_1"
maxOccurs="unbounded"/>
      <xsd:element name="language_2_2" type="languageType_2_2"
maxOccurs="unbounded"/>
      <xsd:element name="language_2_3" type="languageType_2_3"
maxOccurs="unbounded"/>
      <xsd:element name="language_3_1" type="languageType_3_1"
maxOccurs="unbounded"/>
      <xsd:element name="language_3_2" type="languageType_3_2"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

```

```

        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>

```

### 3.2.7 Использование групп элементов и атрибутов

В XML-схемах существует возможность задания групп элементов и атрибутов и их многократного использования.

#### Пример 3.12.

##### Файл XML:

```

<?xml version="1.0" encoding="Windows-1251"?>
<!-- Языки разметки -->
<languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="compl_content_group.xsd">

```

```

    <language>
        <name attr2="FIX" attr3="C" id="id1">HTML</name>
        <year>B</year>
    </language>

```

```

    <language>
        <howold attr3="C" attr1="123" id="id2"
attr2="string">10</howold>
    </language>
</languages>

```

##### Файл XSD:

```

<?xml version="1.0" encoding="Windows-1251"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <!-- Группа атрибутов -->
    <xsd:attributeGroup name="Example_AttributeGroup">

```



```

    <xsd:attribute name="attr1" type="xsd:integer" use="optional"
default="333"/>
    <xsd:attribute name="attr2" type="xsd:string" use="required"
fixed="FIX"/>
    <xsd:attribute name="attr3" type="Example_AttrType"
use="required"/>
</xsd:attributeGroup>

```

С помощью элемента `xsd:attributeGroup` определяется группа атрибутов. Для элемента `xsd:attribute` атрибут `use` может принимать следующие значения:

- `use="required"` – обязательный атрибут.
- `use="optional"` – необязательный атрибут.
- `use="default"` задает значение по умолчанию, если атрибут будет отсутствовать в документе.
- `use="fixed"` задает только одно фиксированное значение для атрибута.

```

<xsd:simpleType name="Example_AttrType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value='A' />
    <xsd:enumeration value='B' />
    <xsd:enumeration value='C' />
  </xsd:restriction>
</xsd:simpleType>

```

Определяется простой тип для использования в качестве значения атрибута.

```

<xsd:complexType name="Example_Element_Type">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attributeGroup ref="Example_AttributeGroup"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

```

Ссылка на группу атрибутов. Атрибуты, определенные в группе, будут добавлены в тип.

```

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

```

В дополнение к группе атрибутов можно отдельно определять атрибуты.

```

        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

<xsd:group name="Example_group">
    <xsd:sequence>
        <xsd:element name="name" type="Example_Element_Type"/>
        <xsd:element name="year" type="Example_AttrType"/>
    </xsd:sequence>
</xsd:group>

```

**Объявление группы элементов.**

```

<!-- Может повторяться вся последовательность -->
<xsd:complexType name="languageType">
    <xsd:choice>
        <!-- Группа может встречаться один и более раз -->
        <xsd:group ref="Example_group" minOccurs="1"
maxOccurs="unbounded"/>

```

Ссылка на группу элементов. Элементы, определенные в группе, будут добавлены в тип.

```

        <xsd:element name="howold" type="Example_Element_Type"/>
    </xsd:choice>
</xsd:complexType>

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

```

```

    </xsd:element>
</xsd:schema>

```

### 3.2.8 Использование смешанной модели содержимого элемента

Смешанная модель содержимого элемента предполагает, что в элемент могут быть вложены и текст, и другие элементы. Пример:

```

<language>
    Текст
    <y>01.01.1986</y>
    Текст
    <n>SGML</n>
</language>

```

Для определения смешанной модели в элементе `xsd:complexType` необходимо указать атрибут `mixed="true"`.

#### Пример 3.13.

##### Файл XML:

```

<?xml version="1.0" encoding="Windows-1251"?>
<!-- Языки разметки -->
<languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="compl_content_mix.xsd">

    <language_sequence>текст<name>HTML</name>текст<year>01.01.1990<
/year>текст<howold>14</howold>текст
    </language_sequence>

    <language_choice>текст<name>XML</name>текст
    </language_choice>

    <language_all>текст<year>01.01.1986</year>текст<howold>10</howo
ld>текст<name>SGML</name>текст
    </language_all>
</languages>

```

**Файл XSD:**

```
<?xml version="1.0" encoding="Windows-1251"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- Последовательность элементов -->
  <xsd:complexType name="languageType_sequence" mixed="true">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="year" type="xsd:string"/>
      <xsd:element name="howold" type="xsd:integer"/>
    </xsd:sequence>
  </xsd:complexType>
```

В элементе `xsd:complexType` атрибут `mixed="true"` определяет смешанную модель.

```
<!-- Выбор элементов -->
<xsd:complexType name="languageType_choice" mixed="true">
  <xsd:choice>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="year" type="xsd:string"/>
    <xsd:element name="howold" type="xsd:integer"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="languageType_all" mixed="true">
  <xsd:all>
    <xsd:element name="name" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
    <xsd:element name="year" type="xsd:string"/>
    <xsd:element name="howold" type="xsd:integer"/>
  </xsd:all>
</xsd:complexType>

<xsd:element name="languages">
```

[Оглавление](#)

```

<xsd:complexType>
  <xsd:sequence>
    <xsd:element name="language_sequence"
type="languageType_sequence"/>
    <xsd:element name="language_choice"
type="languageType_choice"/>
    <xsd:element name="language_all"
type="languageType_all"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>

```

### 3.2.9 Использование «nil» для пустых элементов

Атрибут `xsi:nil` позволяет определять пустое значение элементов в XML-документе. В XML-схеме для того, чтобы указать, что элемент может иметь пустые значения, используется `nillable="true"` при объявлении элемента.

#### Пример 3.14.

##### Файл XML:

```

<?xml version="1.0" encoding="Windows-1251"?>
<!-- Языки разметки -->
<languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="compl_content_nil1.xsd">
  <language>
    <name>HTML</name>
    <name>HTML</name>

    <name xsi:nil="true"/>

```

Элемент `name` не имеет содержимого. Атрибут `xsi:nil="true"` указывает, что элемент имеет пустое значение.

Обратите внимание, что атрибут `xsi:nil` (префикс `xsi`) относится к пространству имен, через которое к XML-документу подключается XML-схема.

```

    <year>01.01.1986</year>
    <year>01.01.1986</year>
    <year xsi:nil="true"/>

    <howold>10</howold>
    <howold>10</howold>
    <howold xsi:nil="true"></howold>
  </language>
</languages>

```

### Файл XSD:

```

<?xml version="1.0" encoding="Windows-1251"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="languageType">
    <xsd:choice minOccurs="1" maxOccurs="unbounded">
      <xsd:element name="name" type="xsd:string" nillable="true"/>
      <xsd:element name="year" type="xsd:string" nillable="true"/>
      <xsd:element name="howold" type="xsd:integer" nillable="true"/>
    </xsd:choice>
  </xsd:complexType>

  <xsd:element name="languages">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="language" type="languageType"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

Атрибут `nillable="true"` указывает, что элемент `name` может быть пустым.

### 3.2.10 Указание типа элемента в XML-документе

Атрибут `xsi:type` позволяет элементу в документе XML ссылаться на тип данных в схеме. При этом схема состоит только из типов, которые не связаны с элементами.

#### Пример 3.15.

##### Файл XML:

```
<?xml version="1.0" encoding="Windows-1251"?>
<!-- Языки разметки -->
<languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="compl_content_type.xsd">
    <language xsi:type="languageType">
```

Тип элемента указывается явно с помощью атрибута `xsi:type`.

Обратите внимание, что атрибут `xsi:type` (префикс `xsi`) относится к пространству имен, через которое к XML-документу подключается XML-схема.

```
        <name>HTML</name>
        <name xsi:nil="true"/>
        <year>01.01.1986</year>
        <howold>10</howold>
    </language>
</languages>
```

##### Файл XSD:

```
<?xml version="1.0" encoding="Windows-1251"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:complexType name="languageType">
    <xsd:choice minOccurs="1" maxOccurs="unbounded">
        <xsd:element name="name" type="xsd:string" nillable="true"/>
        <xsd:element name="year" type="xsd:string" nillable="true"/>
        <xsd:element name="howold" type="xsd:integer" nillable="true"/>
    </xsd:choice>
```

```
</xsd:complexType>
```

Объявленный тип не связан ни с одним из элементов. Привязка осуществляется в XML-документе.

```
<xsd:element name="languages" type="xsd:anyType"/>
</xsd:schema>
```

Элемент `languages` может быть произвольного типа (`xsd:anyType`), то есть его содержимое никак не контролируется.

### 3.2.11 Использование аннотаций

Аннотации (`xsd:annotation`) позволяют документировать XML-схему. В элемент `xsd:annotation` могут быть вложены два элемента:

- Элемент `xsd:documentation`, который позволяет добавлять комментарии в XML-схему.
- Элемент `xsd:appinfo`, который позволяет добавлять информацию для программ, обрабатывающих XML-схемы.

В качестве примера такой программы можно назвать «Schematron». Это надстройка над XML-схемой, которая позволяет осуществлять дополнительные проверки зависимостей между элементами.

Аннотация может быть вложена практически в любой элемент схемы, например в `xsd:schema`, `xsd:complexType`, `xsd:element`.

#### Пример 3.16.

##### Файл XSD с аннотациями:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:documentation xml:lang="ru">Комментарий</xsd:documentation>
    <xsd:appinfo>Информация для программы, которая работает с
аннотациями</xsd:appinfo>
  </xsd:annotation>
```



```

<xsd:complexType name="languageType">
  <xsd:annotation>
    <xsd:documentation xml:lang="ru"> Комментарий
  </xsd:documentation>
  </xsd:annotation>

  <xsd:sequence>
    <xsd:annotation>
      <xsd:documentation xml:lang="ru"> Комментарий
    </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="year" type="xsd:string">
      <xsd:annotation>
        <xsd:documentation xml:lang="ru"> Комментарий
      </xsd:documentation>
    </xsd:annotation>
    </xsd:element>
    <xsd:element name="howold" type="xsd:integer"/>
  </xsd:sequence>
  <xsd:attribute name="id" use="required" type="xsd:string">
    <xsd:annotation>
      <xsd:documentation xml:lang="ru"> Комментарий
    </xsd:documentation>
  </xsd:annotation>
  </xsd:attribute>
</xsd:complexType>

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

```

```

    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

### 3.2.12 Задание ключей и уникальности

В XML-схемах существует способ проверки связей между элементами и проверки уникальности, который похож на первичные, вторичные ключи и уникальные индексы в реляционных базах данных.

Элементы, используемые для задания ключей и уникальности:

- `xsd:key` – «первичный ключ».
- `xsd:keyref` – «вторичный ключ».
- `xsd:unique` – уникальная комбинация значений. Отличие от `xsd:key` в том, что проверяемые значения могут быть пустыми.

В каждый из этих элементов должны быть вложены:

- `xsd:selector` – содержит XPath-выражение для определения уникальности.
- `xsd:field` – содержит XPath-выражение для задания поля в селекторе.

Ограничение задается на том уровне вложенности, на котором объявляются элементы `xsd:key`, `xsd:keyref`, `xsd:unique`. Поэтому для того, чтобы задать ограничение в рамках всего документа, эти элементы должны быть объявлены в корневом элементе документа.

#### Пример 3.17.

##### Файл XML:

```

<?xml version="1.0" encoding="utf-8"?>
<Institute xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="compl_content_key.xsd">
  <!-- Список курсов -->
  <Courses>
    <Course>
      <CourseId>1</CourseId>
    </Course>
  </Courses>
</Institute>

```

```

    <CourseName>Интернет-технологии</CourseName>
</Course>
<Course>
    <CourseId>2</CourseId>
    <CourseName>Язык XML</CourseName>
</Course>
</Courses>

```

```

<!-- Список кафедр -->

```

```

<Departments>
    <Department>
        <DepartmentId>1</DepartmentId>
        <DepartmentName>ИУ-5</DepartmentName>
    </Department>
    <Department>
        <DepartmentId>2</DepartmentId>
        <DepartmentName>ИУ-7</DepartmentName>
    </Department>
</Departments>

```

```

<!-- Учебный план -->

```

```

<PlanElements>
    <PlanElement>
        <!-- Курс -->
        <CourseId>1</CourseId>
        <!-- Кафедра -->
        <DepartmentId>1</DepartmentId>
        <!-- Семестр -->
        <Semestr>3</Semestr>
    </PlanElement>
    <PlanElement>
        <CourseId>1</CourseId>
        <DepartmentId>2</DepartmentId>
        <Semestr>4</Semestr>
    </PlanElement>

```

```

<PlanElement>
  <CourseId>2</CourseId>
  <DepartmentId>1</DepartmentId>
  <Semestr>5</Semestr>
</PlanElement>
<PlanElement>
  <CourseId>2</CourseId>
  <DepartmentId>2</DepartmentId>
  <Semestr>5</Semestr>
</PlanElement>
</PlanElements>
</Institute>

```

### Файл XSD:

```

<?xml version="1.0" encoding="Windows-1251"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- Предмет -->
  <xsd:element name="Course">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="CourseId" type="xsd:integer"/>
        <xsd:element name="CourseName" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

```

Обратите внимание, что в этой схеме все элементы объявлены как глобальные (у элемента `xsd:element` задан атрибут `name`).

Для ссылки на этот элемент внутри другого элемента используется `xsd:element` с атрибутом `ref`.

```

<!-- Кафедра -->
<xsd:element name="Department">
  <xsd:complexType>

```

```

    <xsd:sequence>
      <xsd:element name="DepartmentId" type="xsd:integer"/>
      <xsd:element name="DepartmentName" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

```

<!-- Учебный план -->

```

```

<xsd:element name="PlanElement">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="CourseId" type="xsd:integer"/>
      <xsd:element name="DepartmentId" type="xsd:integer"/>
      <xsd:element name="Semestr" type="xsd:integer"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

```

<xsd:element name="Institute">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Courses">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="Course" maxOccurs="unbounded"/>

```

Ссылка на элемент Course, который был объявлен ранее. Обратите внимание, что это ссылка не на тип данных, а ссылка на сам элемент по имени элемента.

```

      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
<xsd:element name="Departments">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Department" maxOccurs="unbounded"/>

```

```

        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="PlanElements">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="PlanElement" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>

```

```

<!-- Ключ для предмета -->
<xsd:key name="KeyCourseId">
    <xsd:selector xpath="./Course"/>
    <xsd:field xpath="CourseId"/>
</xsd:key>

```

Определение первичного ключа с названием KeyCourseId. В элементе Course, который определяется XPath-выражением (./Course) вложенный элемент CourseId должен быть уникальным.

```

<!-- Ключ для кафедры -->
<xsd:key name="KeyDepartmentId">
    <xsd:selector xpath="./Department"/>
    <xsd:field xpath="DepartmentId"/>
</xsd:key>

```

Первичный ключ для кафедры.

```

<!-- Вторичные ключи для предмета и кафедры -->
<xsd:keyref name="RefKeyCourseId" refer="KeyCourseId">
    <xsd:selector xpath="./PlanElement"/>
    <xsd:field xpath="CourseId"/>
</xsd:keyref>

```

Определение вторичного ключа с названием RefKeyCourseId. Элемент CourseId, вложенный в элемент PlanElement, который определяется XPath-выражением (./PlanElement), должен входить в список значений первичного ключа с названием KeyCourseId. То есть во вторичном ключе не может встретиться такое значение CourseId, которого нет в первичном ключе.

```
<xsd:keyref name="RefKeyDepartmentId"
refer="KeyDepartmentId">
  <xsd:selector xpath="./PlanElement"/>
  <xsd:field xpath="DepartmentId"/>
</xsd:keyref>
```

Определение вторичного ключа для кафедры.

```
<xsd:unique name="UniquePlanElement">
  <xsd:selector xpath="./PlanElement"/>
  <xsd:field xpath="CourseId"/>
  <xsd:field xpath="DepartmentId"/>
  <xsd:field xpath="Semestr"/>
</xsd:unique>
```

В элементе PlanElement, который определяется XPath-выражением (./PlanElement), комбинация вложенных элементов CourseId, DepartmentId и Semestr должна быть уникальна. То есть уникальна комбинация курса, кафедры и семестра в учебном плане.

```
</xsd:element>
</xsd:schema>
```

### 3.2.13 Использование пространств имен

XML-схема позволяет проверять действительность документов, содержащих пространства имен.

Пример простого документа и схемы с проверкой пространства имен.

### Пример 3.18.

#### Файл XML:

```
<?xml version="1.0" encoding="Windows-1251"?>
<q:languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ru/namespace-example
ns_example_1.xsd" xmlns:q="http://www.ru/namespace-example">
```

В случае использования пространств имен для подключения схемы используется не атрибут `noNamespaceSchemaLocation`, а атрибут `schemaLocation`.

Значением атрибута является список из двух значений. Первое значение – название пространства имен (в нашем примере `http://www.ru/namespace-example`), второе значение – URI файла схемы (в нашем примере `ns_example_1.xsd`).

Так как в схеме объявлено `elementFormDefault="unqualified"`, то префикс пространства имен должен быть указан только у корневого элемента XML-документа.

Так как в схеме объявлено `attributeFormDefault="unqualified"`, то префикс пространства имен у атрибутов не объявляется.

Префикс пространства имен в XML-документе « q » не совпадает с префиксом пространства имен в XML-схеме « ns1 ». Названия префиксов не имеют значения, должны совпадать значения пространства имен «`http://www.ru/namespace-example`».

```
<language id="123">
  <name>HTML</name>
  <year>01.01.1990</year>
  <howold>15</howold>
</language>
</q:languages>
```

#### Файл XSD:

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

[Оглавление](#)



```
xmlns:ns1="http://www.ru/namespace-example"
targetNamespace="http://www.ru/namespace-example"
elementFormDefault="unqualified"
attributeFormDefault="unqualified">
```

Атрибут `xmlns:ns1="http://www.ru/namespace-example"` содержит объявление пространства имен. «ns1» – префикс пространства имен, «http://www.ru/namespace-example» – значение пространства имен.

Атрибут `targetNamespace="http://www.ru/namespace-example"` определяет пространство имен для проверяемого документа.

Атрибуты `elementFormDefault = "unqualified"` и `attributeFormDefault = "unqualified"` означают, что не требуется явно указывать префикс пространства имен для элементов и атрибутов XML-документа.

```
<!-- Определение типа -->
<xsd:simpleType name="howoldType">
  <xsd:restriction base="xsd:integer">
    <xsd:enumeration value='10' />
    <xsd:enumeration value='15' />
    <xsd:enumeration value='20' />
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="languageType">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="year" type="xsd:string"/>
    <xsd:element name="howold" type="ns1:howoldType"/>
```

При ссылке на типы, объявленные в схеме с пространством имен, необходимо явно ссылаться на пространство имен, указывая префикс (ns1:howoldType).

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

```

</xsd:complexType>

<xsd:complexType name="languagesType">
  <xsd:sequence>
    <xsd:element name="language" type="ns1:languageType"
maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="languages" type="ns1:languagesType"/>
</xsd:schema>

```

### Пример 3.19.

Если в XML-схеме (xsd:schema) указано elementFormDefault="qualified", то требуется явно указывать префикс пространства имен для всех элементов в XML-документе.

```

<?xml version="1.0" encoding="Windows-1251"?>
<q:languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ru/namespace-example
ns_example_ge.xsd" xmlns:q="http://www.ru/namespace-example">
  <q:language id="123">
    <q:name>HTML</q:name>
    <q:year>01.01.1990</q:year>
    <q:howold>15</q:howold>
  </q:language>
</q:languages>

```

Или префикс может быть совсем не указан, если указано пространство имен по умолчанию. Тогда вместо атрибута xmlns:q="http://www.ru/namespace-example" необходимо указывать атрибут xmlns="http://www.ru/namespace-example".

```

<?xml version="1.0" encoding="Windows-1251"?>
<languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ru/namespace-example
ns_example_ge.xsd" xmlns="http://www.ru/namespace-example">

```

```

<language id="123">
  <name>HTML</name>
  <year>01.01.1990</year>
  <howold>15</howold>
</language>
</languages>

```

### Пример 3.20.

Если в XML-схеме (xsd:schema) указано `elementFormDefault="unqualified"` и `attributeFormDefault="qualified"`, то не требуется явно указывать префикс пространства имен для элементов, но требуется явно указывать префикс пространства имен для атрибутов в XML-документе.

```

<?xml version="1.0" encoding="Windows-1251"?>
<q:languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ru/namespace-example
ns_example_qa_1.xsd" xmlns:q="http://www.ru/namespace-example">
  <language q:id="123">
    <name>HTML</name>
    <year>01.01.1990</year>
    <howold>15</howold>
  </language>
</q:languages>

```

Обратите внимание, что в атрибуте `q:id` указан префикс пространства имен.

Если в схеме объявлено `elementFormDefault="qualified"` и `attributeFormDefault="qualified"`, то префикс пространства имен должен быть указан у всех элементов и атрибутов.

```

<q:languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ru/namespace-example
ns_example_qa_2.xsd" xmlns:q="http://www.ru/namespace-example">
  <q:language q:id="123">
    <q:name>HTML</q:name>
    <q:year>01.01.1990</q:year>
  </q:language>
</q:languages>

```

```

        <q:howold>15</q:howold>
    </q:language>
</q:languages>

```

Или префикс может быть совсем не указан, если указано пространство имен по умолчанию. Тогда вместо атрибута `xmlns:q="http://www.ru/namespace-example"` необходимо указывать атрибут `xmlns="http://www.ru/namespace-example"`.

```

<?xml version="1.0" encoding="Windows-1251"?>
<languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ru/namespace-example
ns_example_qa_2.xsd" xmlns="http://www.ru/namespace-example">
    <language id="123">
        <name>HTML</name>
        <year>01.01.1990</year>
        <howold>15</howold>
    </language>
</languages>

```

С использованием атрибута `form="qualified"` можно явно указывать квалифицированность отдельных элементов и атрибутов в документе.

### Пример 3.21.

#### Файл XML:

```

<?xml version="1.0" encoding="Windows-1251"?>
<q:languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ru/namespace-example
ns_example_form.xsd" xmlns:q="http://www.ru/namespace-example">
    <language q:id="123">
        <q:name>HTML</q:name>
        <year>01.01.1990</year>
        <howold>15</howold>
    </language>
</q:languages>

```

**Файл XSD:**

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ns1="http://www.ru/namespace-example"
            targetNamespace="http://www.ru/namespace-example"
            elementFormDefault="unqualified"
            attributeFormDefault="unqualified">

    <!-- Определение типа -->
    <xsd:simpleType name="howoldType">
        <xsd:restriction base="xsd:integer">
            <xsd:enumeration value='10' />
            <xsd:enumeration value='15' />
            <xsd:enumeration value='20' />
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:complexType name="languageType">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string" form="qualified"/>

```

Элемент name должен быть квалифицированным (должен быть явно указан префикс пространства имен).

```

            <xsd:element name="year" type="xsd:string"/>
            <xsd:element name="howold" type="ns1:howoldType"/>
        </xsd:sequence>
        <xsd:attribute name="id" use="required" type="xsd:string"
form="qualified"/>

```

Атрибут id должен быть квалифицированным.

```

</xsd:complexType>
<xsd:complexType name="languagesType">
    <xsd:sequence>

```

```

        <xsd:element name="language" type="ns1:languageType"
maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="languages" type="ns1:languagesType"/>
</xsd:schema>

```

### 3.2.14 Создание XML-схем, состоящих из нескольких файлов

Элемент `xsd:include` позволяет включать схему из внешнего файла. В схемах должны использоваться одинаковые пространства имен.

#### Пример 3.22.

##### Файл XML:

```

<?xml version="1.0" encoding="Windows-1251"?>
<languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="include.xsd">
    <language id="123">
        <name>HTML</name>
        <year>01.01.1990</year>
        <howold>15</howold>
    </language>
</languages>

```

##### Файл include.xsd:

```

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

```

```

    <xsd:include schemaLocation="include_inner.xsd"/>

```

Включение файла, содержащего часть XML-схемы.

```

<xsd:complexType name="languageType">
    <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>

```

```

        <xsd:element name="year" type="xsd:string"/>
        <xsd:element name="howold" type="howoldType"/>
    </xsd:sequence>
    <xsd:attribute name="id" use="required" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="languagesType">
    <xsd:sequence>
        <xsd:element name="language" type="languageType"
maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="languages" type="languagesType"/>
</xsd:schema>

```

### Файл `include_inner.xsd`:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!-- Перечисление вынесено в отдельную схему -->
    <xsd:simpleType name="howoldType">
        <xsd:restriction base="xsd:integer">
            <xsd:enumeration value='10' />
            <xsd:enumeration value='15' />
            <xsd:enumeration value='20' />
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>

```

Импортирование схем применяется при использовании нескольких пространств имен.

### Пример 3.23.

#### Файл XML:

```

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

```

```

<q:languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ru/namespace-example import.xsd"
xmlns:q="http://www.ru/namespace-example" xmlns:ns2="www.ns2.ru">
  <language id="123">
    <name>HTML</name>
    <year>01.01.1990</year>
    <ns2:howold>15</ns2:howold>
  </language>
</q:languages>

```

### Файл import.xsd:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.ru/namespace-example"
xmlns:ns1="http://www.ru/namespace-example" xmlns:ns2="www.ns2.ru"
elementFormDefault="unqualified" attributeFormDefault="unqualified">

  <xsd:import namespace="www.ns2.ru"
schemaLocation="import_inner.xsd"/>

```

**Вставка внешней схемы, в которой используется другое пространство имен.**

```

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

    <xsd:element ref="ns2:howold"/>

```

**xsd:element** ссылается на импортированное объявление элемента ns2:howold.

```

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

```



```

        <xsd:element name="language" type="ns1:languageType"
maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
    <xsd:element name="languages" type="ns1:languagesType"/>
</xsd:schema>

```

### Файл `import_inner.xsd`:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns2="www.ns2.ru" targetNamespace="www.ns2.ru"
elementFormDefault="unqualified" attributeFormDefault="unqualified">
    <!-- Перечисление вынесено в отдельную схему -->
    <xsd:simpleType name="howoldType">
        <xsd:restriction base="xsd:integer">
            <xsd:enumeration value='10' />
            <xsd:enumeration value='15' />
            <xsd:enumeration value='20' />
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:element name="howold" type="ns2:howoldType"/>
</xsd:schema>

```

С помощью элемента `xsd:any` в XML-документ можно вставлять любые элементы, принадлежащие определенному пространству имен.

### Пример 3.24.

#### Файл XML:

```

<?xml version="1.0" encoding="Windows-1251"?>
<languages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="any.xsd"
xmlns:ns1="http://www.ns1.ru">
    <language id="123">
        <name>HTML</name>
        <year>01.01.1990</year>
    </language>
</languages>

```

```

    <ns1:AnyElement>
        <B>qwerty</B>
    </ns1:AnyElement>

</language>
</languages>

```

### Файл XSD:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="languageType">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="year" type="xsd:string"/>

            <xsd:any namespace="http://www.ns1.ru"
processContents="skip" minOccurs="0" maxOccurs="unbounded"/>

```

В этом месте последовательности могут следовать любые элементы пространства имен «http://www.ns1.ru». Атрибут processContents="skip" указывает, что элементы, принадлежащие этому пространству имен, не нужно проверять на действительность.

```

        </xsd:sequence>
        <xsd:attribute name="id" use="required" type="xsd:string"/>
    </xsd:complexType>
    <xsd:complexType name="languagesType">
        <xsd:sequence>
            <xsd:element name="language" type="languageType"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="languages" type="languagesType"/>
</xsd:schema>

```

### 3.2.15 Шаблоны проектирования схем

Так как в XML-схемах существуют различные варианты совместного использования составных типов и элементов, то появилось понятие «шаблон (паттерн) проектирования схем».

Существует четыре наиболее распространенных шаблона проектирования схем:

- Venetian Blind (венецианская штора).
- Russian Doll (матрешка).
- Salami Slice (ломтики салями).
- Garden of Eden (Райский Сад, идеальный шаблон).

Некоторые средства разработки (в частности NetBeans) умеют преобразовывать схемы из одного шаблона в другой.

Для преобразования схем в соответствии со следующими примерами необходимо установить платформу Java - JDK (<http://www.oracle.com/technetwork/java/javase/downloads/index.html?ssSourceSiteId=otnjp>) и среду разработки NetBeans (<https://netbeans.org/>).

Пакет преобразования схем не установлен в NetBeans по умолчанию, рекомендации по его установке указаны в статье разработчика ([https://blogs.oracle.com/geertjan/entry/xml\\_schema\\_editor\\_in\\_netbeans](https://blogs.oracle.com/geertjan/entry/xml_schema_editor_in_netbeans)).

Использование NetBeans для автоматизированного применения шаблонов проектирования указано в документации по NetBeans (<http://www.oracle.com/technetwork/java/design-patterns-142138.html>).

Рассмотрим шаблоны более подробно.

В шаблоне Venetian Blind (венецианская штора) определен один глобальный элемент. Остальные вложены в глобальный с использованием именованных составных типов и групп. Составные типы и группы могут быть использованы несколько раз.

То есть в схеме определены глобальные типы `xsd:complexType` и `xsd:simpleType` с атрибутом `name`. Определен единственный глобальный элемент `xsd:element`, который ссылается на глобальные типы с помощью атрибута `type`.

Данный шаблон является наиболее распространенным. Все предыдущие примеры в этой главе (за исключением примера на ключи) разработаны именно по этому шаблону.

### Пример 3.25.

#### Файл XSD:

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

  <!-- Определение типа для атрибута -->
  <xsd:simpleType name="howoldType">
    <xsd:restriction base="xsd:integer">
      <xsd:enumeration value='10' />
      <xsd:enumeration value='15' />
      <xsd:enumeration value='20' />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="languageType">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="year" type="xsd:string"/>
      <xsd:element name="howold" type="howoldType"/>
    </xsd:sequence>
    <xsd:attribute name="id" use="required" type="xsd:string"/>
  </xsd:complexType>

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

```

```

        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>

```

### Форма преобразования шаблона в NetBeans:

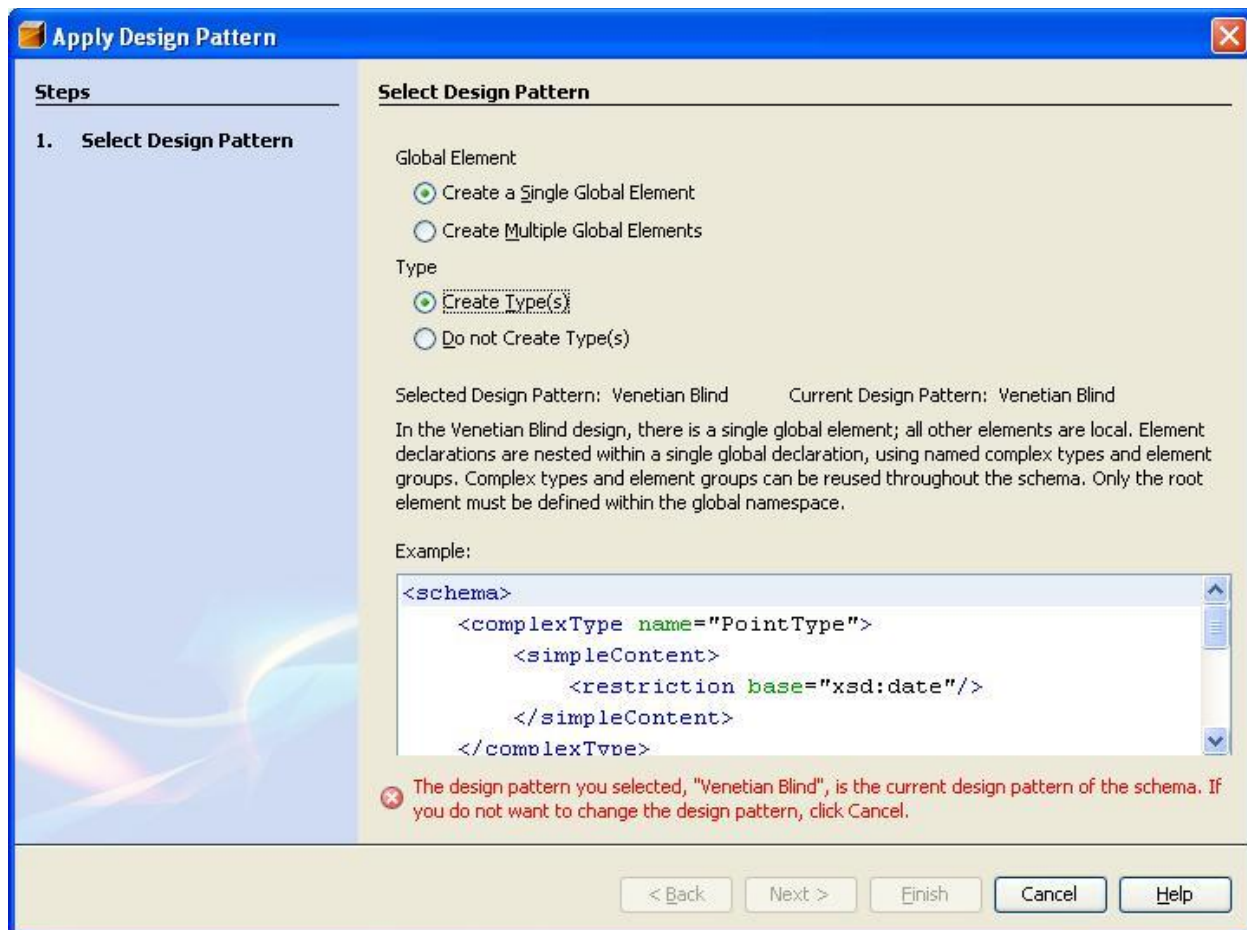


Рис. 3.10. Преобразование XML-схемы в шаблон «венецианская штора».

В шаблоне Russian Doll (матрешка) определен один глобальный элемент. Остальные вложены в глобальный с использованием неименованных составных типов. Составные типы могут быть использованы только один раз.

### Пример 3.26.

#### Файл XSD:

```

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

```

```

<xsd:element name="languages">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="language" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="year" type="xsd:string"/>
            <xsd:element name="howold">
              <xsd:simpleType>
                <xsd:restriction base="xsd:integer">
                  <xsd:enumeration value="10"/>
                  <xsd:enumeration value="15"/>
                  <xsd:enumeration value="20"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:element>
          </xsd:sequence>
          <xsd:attribute name="id" use="required" type="xsd:string"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>

```

### **Форма преобразования шаблона в NetBeans:**

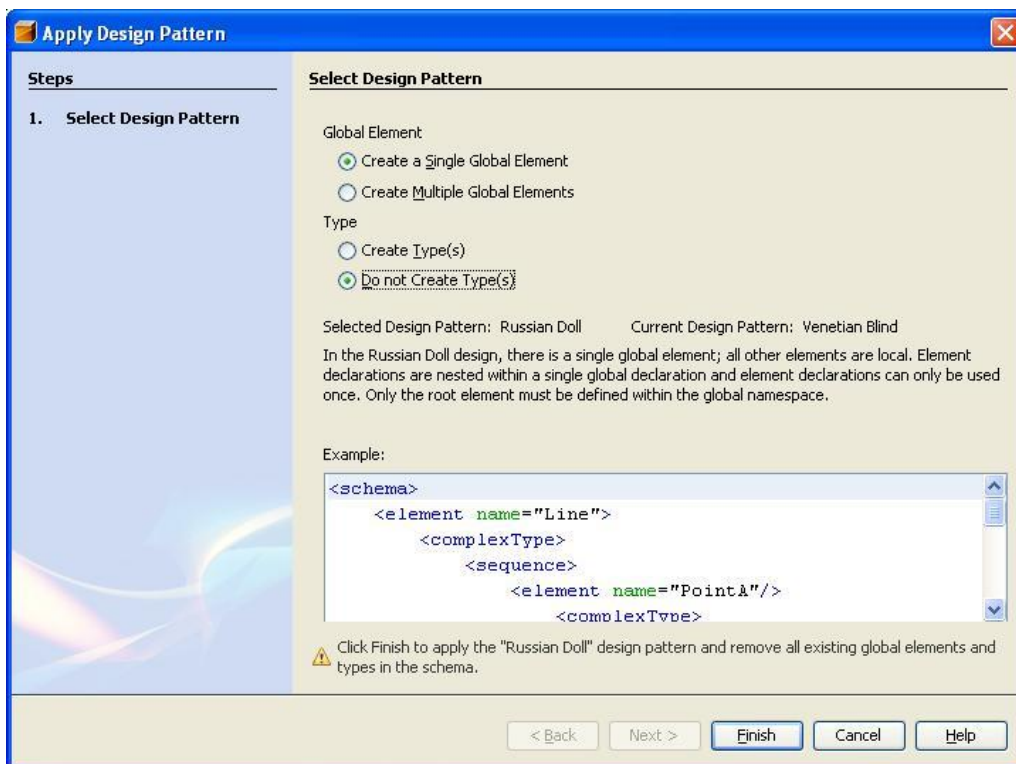


Рис. 3.11. Преобразование XML-схемы в шаблон «матрешка».

В шаблоне Salami Slice (ломтики салями) все элементы объявлены глобальными. Типы вложены в элементы. Используются ссылки на элементы (атрибут ref).

### Пример 3.27.

#### Файл XSD:

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

  <xsd:element name="languages">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="language" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="language">
    <xsd:complexType>
```

```

        <xsd:sequence>
            <xsd:element ref="name"/>
            <xsd:element ref="year"/>
            <xsd:element ref="howold"/>
        </xsd:sequence>
        <xsd:attribute name="id" use="required"
type="xsd:string"/>
    </xsd:complexType>
</xsd:element>

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

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

<xsd:element name="howold">
    <xsd:simpleType>
        <xsd:restriction base="xsd:integer">
            <xsd:enumeration value="10"/>
            <xsd:enumeration value="15"/>
            <xsd:enumeration value="20"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
</xsd:schema>

```

## Форма преобразования шаблона в NetBeans:



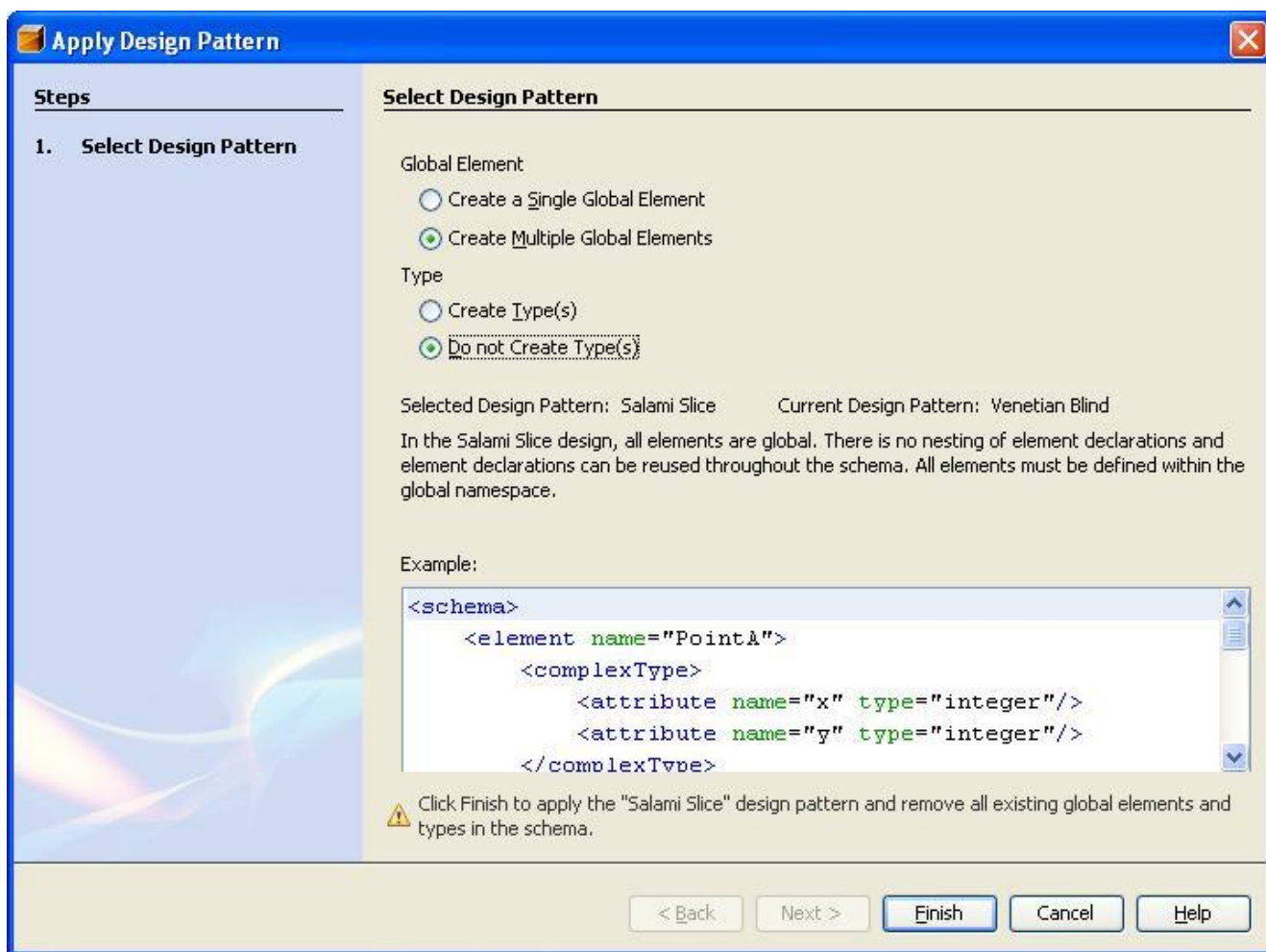


Рис. 3.12. Преобразование XML-схемы в шаблон «ломтики салями».

Шаблон Garden of Eden (Райский Сад, оптимальный шаблон) является комбинацией шаблонов Venetian Blind и Salami Slice. Элементы и типы объявляются глобальными, ссылки создаются по мере необходимости.

### Пример 3.28.

#### Файл XSD:

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

  <xsd:element name="languages" type="languagesType"/>

  <xsd:complexType name="languagesType">
    <xsd:sequence>
      <xsd:element ref="language" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

```

        </xsd:sequence>
    </xsd:complexType>

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

    <xsd:complexType name="languageType">
        <xsd:sequence>
            <xsd:element ref="name"/>
            <xsd:element ref="year"/>
            <xsd:element ref="howold"/>
        </xsd:sequence>
        <xsd:attribute name="id" use="required" type="xsd:string"/>
    </xsd:complexType>

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

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

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

    <xsd:simpleType name="NewHowoldTypeType">
        <xsd:restriction base="xsd:integer">
            <xsd:enumeration value="10"/>
            <xsd:enumeration value="15"/>
            <xsd:enumeration value="20"/>
        </xsd:restriction>
    </xsd:simpleType>

</xsd:schema>

```

### **Форма преобразования шаблона в NetBeans:**

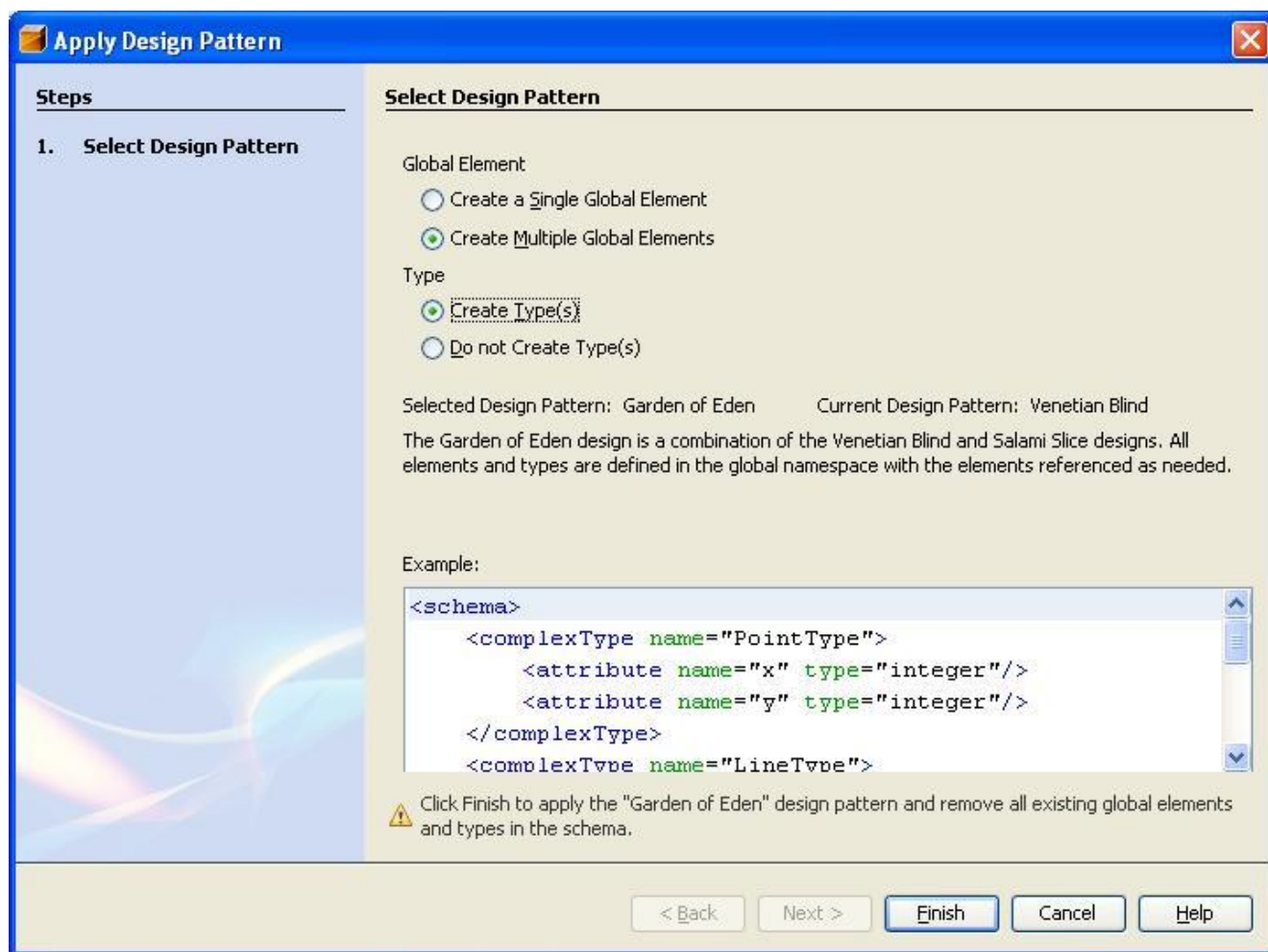


Рис. 3.13. Преобразование XML-схемы в оптимальный шаблон.

### 3.2.16 Профиль XML-схем

Схемы XML являются достаточно сложной спецификацией. Далеко не все возможности схем используются на практике.

Было проведено исследование, в котором изучалось, какие возможности схем используются на практике, а какие нет. Результат исследования назвали профилем XML-схем.

Профиль содержит список наиболее редко используемых возможностей и наиболее часто используемых возможностей.

1. Редко используемые конструкции (встречающиеся в схемах реже, чем в 10% случаев).

- элемент `xsd:all`. Предпочтительно применять вместо него `xsd:choice` или `xsd:sequence`;
- уникальность (Uniqueness): использование элемента `unique` требует, чтобы его содержимое отличалось от остальных элементов в пределах его области видимости. Казалось бы, это удобно, ведь потребность в уникальных идентификаторах вполне естественна. Однако, как выяснилось, в большинстве случаев для таких элементов используются строковые данные. Возможно, уникальность используется на бизнес-уровне передачи данных между системами;
- квалифицированные атрибуты (qualified attributes): использование конструкции `attributeFormDefault="qualified"`. Она не применяется практически ни в одной схеме, хотя многие разработчики используют квалифицированные элементы;
- ключи (keys): использование элементов `key` и `keyref`;
- `nillable`: использование атрибута `nillable`, обуславливающие использование  `xsi:nil`  в экземпляре;
- `mixed`: установка атрибута `mixed="true"` позволяет комбинировать данные и дочерние элементы в одном месте. Разработчики схем четко разделили эти концепции на определенные типы;
- группы (groups): использование `xsd:group` позволяет определять группу для дальнейшего повторного использования. Однако такие элементы чаще всего употребляются с конструкцией `"ref"`, а не добавляются в группы.

2. Часто используемые конструкции (встречаются, по крайней мере, в трети из рассмотренных схем).

- квалифицированные элементы пространства имен: использование конструкции `elementFormDefault="qualified"` для явного задания пространства имен элемента;

- `xsd:sequence`: использование элемента `xsd:sequence`. Это наиболее часто используемый композитор. Он рекомендуется вместо конструкции `xsd:all`, поскольку устраняет двусмысленные модели, и дочерние элементы следуют в определенном порядке;
- расширение `complexType` : создание типа, которые расширяет другой тип, – одна из главных возможностей повторного использования и расширяемости;
- анонимные типы (anonymous types): используются в тех случаях, когда создаваемые типы имеют локальный масштаб и, следовательно, у них отсутствует атрибут `name`. Это бывает очень часто;
- ограничение `simpleType`: производное от `simpleType`, ограничивающее базовый тип;
- перечисления (enumerations): перечень значений – одна из наиболее часто встречающихся конструкций.

### 3.2.17 Существующие стандарты описания XML-схем

Более подробно XML-схемы рассмотрены в спецификации. Спецификация XML-схем состоит из трех частей:

1. **[XML Schema Part 0 Primer, 2004]** – введение в XML-схемы в виде примеров. Существуют неофициальные русские переводы этой части спецификации.

2. **[XML Schema Part 1 Structures, 2004]** – содержит описание конструкций, используемых в XML-схемах.

3. **[XML Schema Part 2 Datatypes, 2004]** – содержит описание типов данных, используемых в XML-схемах.

Кроме рассмотренных XSD-схем, которые являются стандартом веб-консорциума, можно встретить и другие варианты XML-схем.

XDR-схемы (XML Data-Reduced) упрощенный формат описания схем. Появился раньше, чем XSD-схемы, считается устаревшим.

Также существуют RELAX NG схемы, которые имеют более простой синтаксис по сравнению с XSD-схемами.

### **3.3 Материалы для дальнейшего изучения**

Рекомендуется ознакомиться со спецификацией XML-схем [XML Schema Part 0 Primer, 2004] (существует русский перевод), [XML Schema Part 1 Structures, 2004], [XML Schema Part 2 Datatypes, 2004].

### **3.4 Контрольные вопросы**

1. Что такое DTD и для чего используется эта технология?
2. Как в DTD объявляются элементы XML-документа?
3. Как в DTD объявляется последовательность и выбор элементов?
4. Как в DTD объявляется количество вхождений элемента?
5. Как в DTD объявляются атрибуты элементов XML-документа?
6. В чем разница между встроенными и внешними DTD?
7. В чем основное отличие в способах описания содержимого элементов в DTD и XML-схемах?
8. Как присоединить XML-схему к документу XML?
9. Как используются простые типы и ограничения (фасеты) в XML-схемах?
10. Как объявляются списки и объединения в XML-схемах?
11. Как объявляются атрибуты элементов в XML-схемах?
12. Как объявляются сложные (составные) типы в XML-схемах? В чем отличие от DTD?
13. Как объявляется количество вхождений элемента в XML-схемах? В чем отличие от DTD?
14. Как используются группы элементов и атрибутов в XML-схемах?
15. Как объявляется смешанная модель содержимого элемента в XML-схемах?
16. Как используются аннотации в XML-схемах?

17. Как используются ключи и уникальность в XML-схемах?
18. Как создавать XML-схемы, проверяющие пространства имен? Как проверяются квалифицированные и неквалифицированные элементы и атрибуты?
19. Как создавать XML-схемы, состоящие из нескольких файлов?
20. Что такое шаблоны проектирования XML-схем? В чем их особенности?

## **4 Часть 4. Введение в XML–ориентированные СУБД и язык XQuery**

### **4.1 Естественные и приспособленные XML–ориентированные СУБД**

В настоящее время большинство современных СУБД позволяет работать с данными в формате XML, например, Microsoft SQL Server, Oracle, Cache. Для работы с XML используются различные расширения языка SQL, иногда XQuery. В этих СУБД XML не является основным форматом хранения данных. Такие СУБД называют «XML-enabled Database» – СУБД, приспособленные к использованию XML.

Существуют также специализированные XML-ориентированные СУБД, в которых XML является основным форматом хранения данных. Такие СУБД называют «Native XML Database (NXD)» – естественные (прирожденные) XML СУБД.

Часто NXD СУБД вообще не поддерживают реляционную модель. К таким СУБД можно отнести коммерческую Tamino, а также свободно распространяемые eXist и Sedna.

В таких СУБД основным языком обработки данных, как правило, является XQuery.

Более подробно классификация XML-ориентированных СУБД рассмотрена в статье [Бурэ, 2002].

### **4.2 Основы работы с СУБД «eXist»**

СУБД eXist является «естественной» XML-ориентированной СУБД. Эта СУБД хранит данные только в формате XML и не использует другие форматы хранения данных (например, реляционные таблицы).

Это свободно распространяемый программный продукт. Официальный сайт СУБД «eXist» – <http://exist-db.org/>



Хочется также отметить, что eXist является не только СУБД, но и платформой разработки приложений на основе технологии XQuery, содержит встроенную среду разработки eXide.

Эта особенность позволяет разрабатывать веб-приложения на основе eXist без привлечения каких-либо дополнительных программных средств.

#### **4.2.1 Установка eXist**

На сайте eXist можно найти актуальную версию и инструкцию по установке. Версии обновляются достаточно часто, соответственно меняется инструкция по установке, поэтому в данном пособии инструкция по установке не приводится.

При установке eXist требуется задать пароль администратора.

Для работы с eXist необходимо установить платформу Java - JDK (<http://www.oracle.com/technetwork/java/javase/downloads/index.html?ssSourceSiteId=otnjp>).

Традиционным способом установки является установка исполняемого .jar файла.

В случае необходимости eXist можно пересобрать в виде .war-архива (веб-архива) и установить на веб-сервер Apache Tomcat.

Если eXist установлен на локальном веб-сервере Apache Tomcat, то с ним можно работать как с веб-приложением с использованием браузера. Для обращения к eXist используется URI – `http://localhost:8080/exist`

#### **4.2.2 Приборная панель eXist**

В начале работы с eXist открывается основное меню, оформленное в виде приборной панели (dashboard).

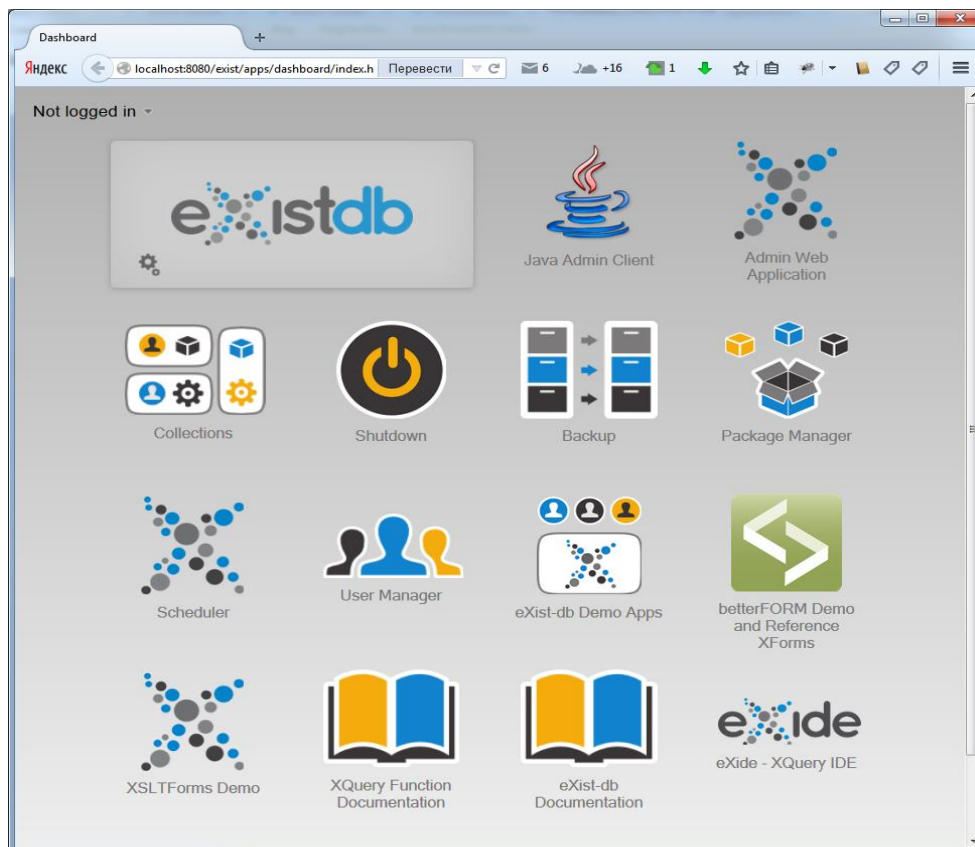


Рис. 4.1. Стартовая страница eXist – приборная панель.

В левом верхнем углу приборной панели расположена гиперссылка для входа в систему.

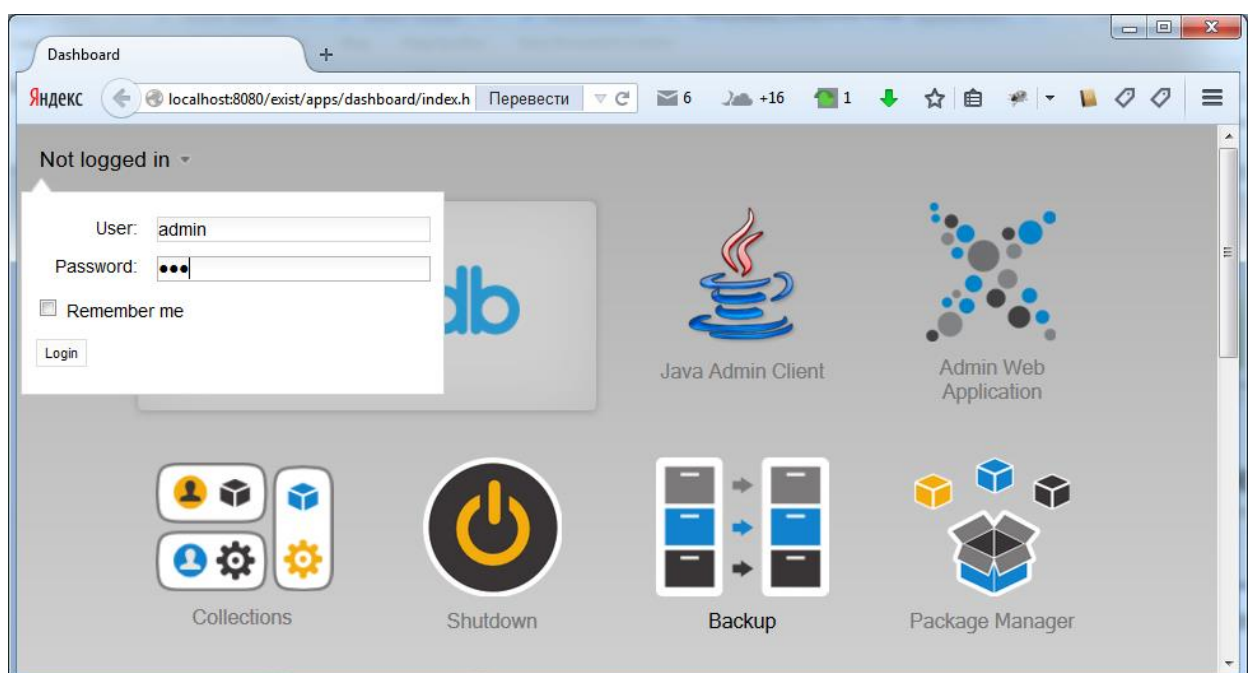


Рис. 4.2. eXist – вход в систему.

По умолчанию в системе существует пользователь «admin» с правами администратора, пароль для данного пользователя указывается при установке eXist.

Приборная панель содержит следующие элементы:

- Java Admin Client – средство администрирования в виде приложения на Java;
- Admin Web application – средство администрирования в виде веб-приложения;
- Collections – средство для работы с коллекциями (рассматривается далее в этой главе);
- Shutdown – завершение работы СУБД;
- Backup – создание и восстановление резервных копий БД;
- Package Manager – работа с пакетами (рассматривается далее в этой главе);
- Sheduler – средство для запуска задач по расписанию;
- User Manager – управление пользователями и правами;
- eXist Demo Apps – примеры веб-приложений, разработанных с использованием eXist;
- betterForm – библиотека для работы с технологиями XForms в виде серверного веб-приложения;
- XSLTForms – другая библиотека для работы с технологиями XForms, которая работает непосредственно в браузере;
- XQuery Functions Documentations – документация по функциям XQuery;
- eXist-db documentation – основная документация eXist;
- eXide – XQuery IDE – среда разработки приложений для eXist, которая работает в браузере (рассматривается далее в этой главе).

### 4.2.3 Работа с пакетами

Система eXist позволяет устанавливать компоненты в виде пакетов с использованием средства Package Manager.

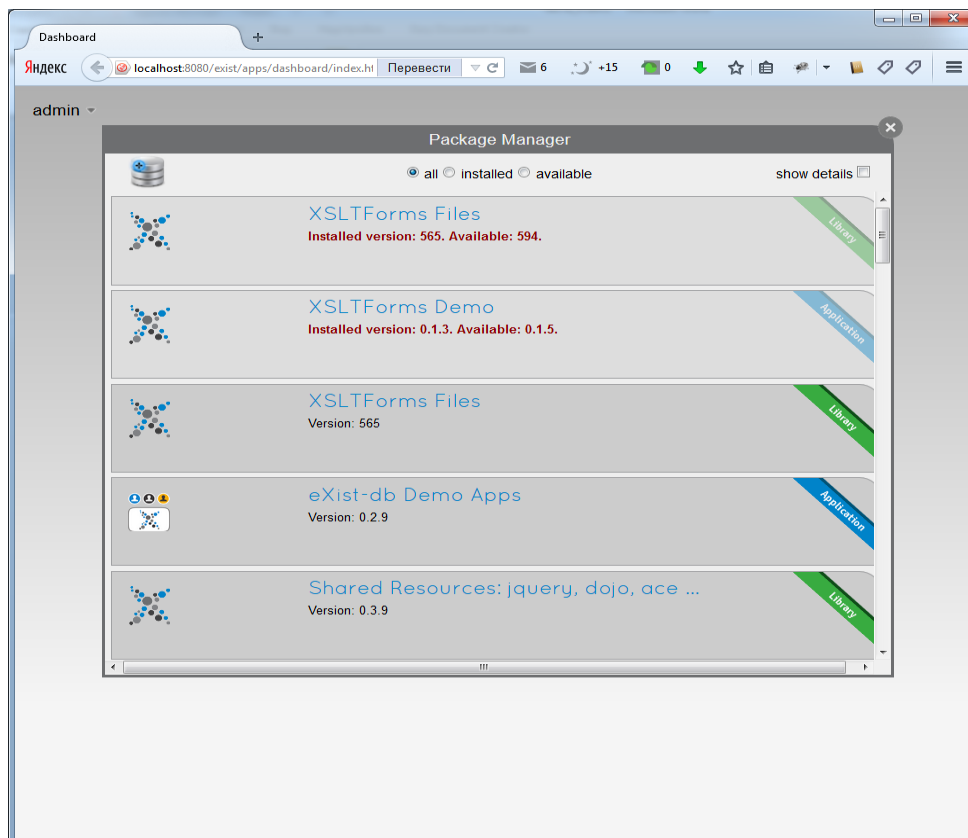


Рис. 4.3. Список компонентов в Package Manager.

Компоненты могут быть библиотеками или приложениями. eXist содержит встроенные средства разработки приложений и возможность их сохранения в виде пакетов. Использование пакетов облегчает перенос разработанных приложений между различными серверами eXist.

### 4.2.4 Среда разработки eXide

Среда разработки eXide (XQuery IDE) позволяет разрабатывать веб-приложения для eXist. eXide работает в браузере и позволяет читать и сохранять данные в eXist.

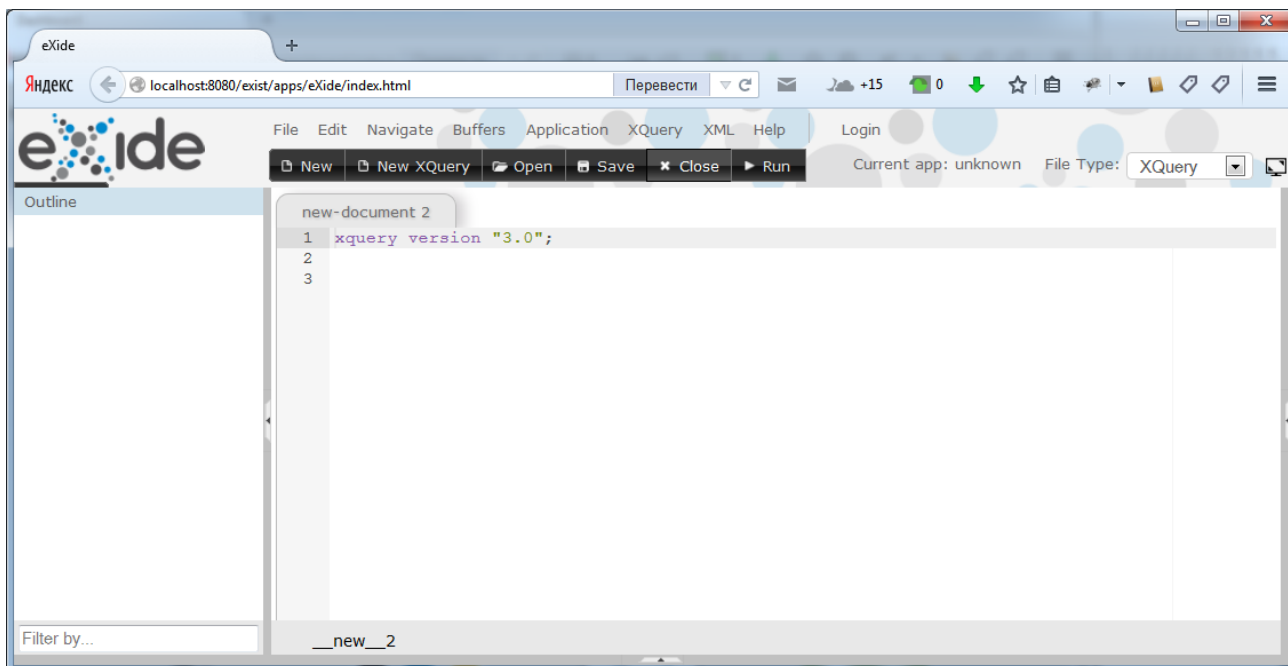


Рис. 4.4. Среда разработки eXide.

eXide содержит редакторы исходного кода для XML, XQuery, HTML, CSS, JavaScript и других языков.

Также eXide поддерживает возможность создания приложений, приложение может быть сохранено в виде пакета и установлено на другой сервер eXist.

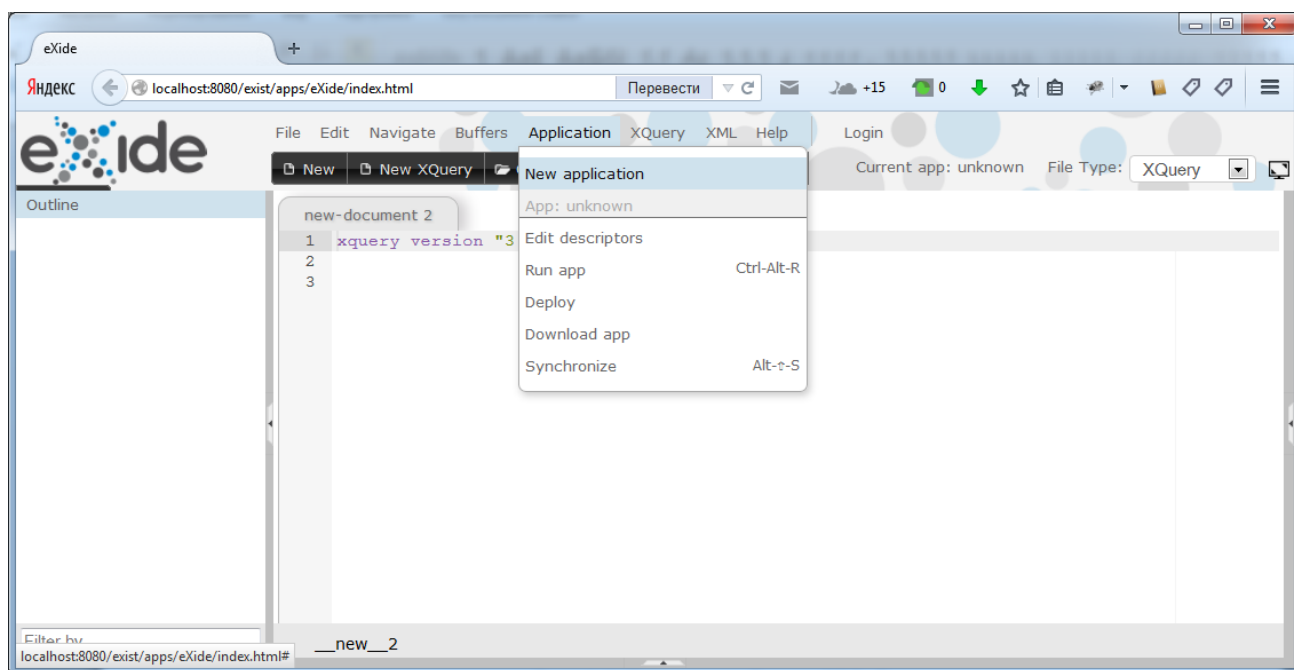


Рис. 4.5. Среда разработки eXide – создание нового приложения.

#### 4.2.5 Работа с коллекциями

Данные хранятся в eXist в каталогах (коллекциях). В каталоге могут находиться подкаталоги, файлы в формате XML. Корневой каталог называется «\db».

Этот вариант хранения данных похож на то, как хранятся файлы в файловой системе. Однако в eXist все данные (каталоги и файлы XML) хранятся в двоичных файлах. За счет этого (как указано в документации eXist) работа с XML-данными происходит быстрее, чем если бы они хранились в файловой системе в виде файлов.

Интересной особенностью eXist является то, что в коллекции можно хранить файлы любых типов. Если в коллекции хранится XML-файл, то его можно обрабатывать с использованием XQuery-сценариев, фактически это аналог таблицы базы данных. Если в коллекции хранится XQuery-сценарий, то его можно вызывать как серверный сценарий через HTTP. Если в коллекции хранится бинарный файл (например, изображение), то к нему можно также обратиться через HTTP (например, показать как картинку из сгенерированной HTML-страницы). Эта особенность позволяет хранить веб-приложение целиком в коллекции eXist.

Для работы с коллекциями и файлами используется элемент приборной панели «Collections».

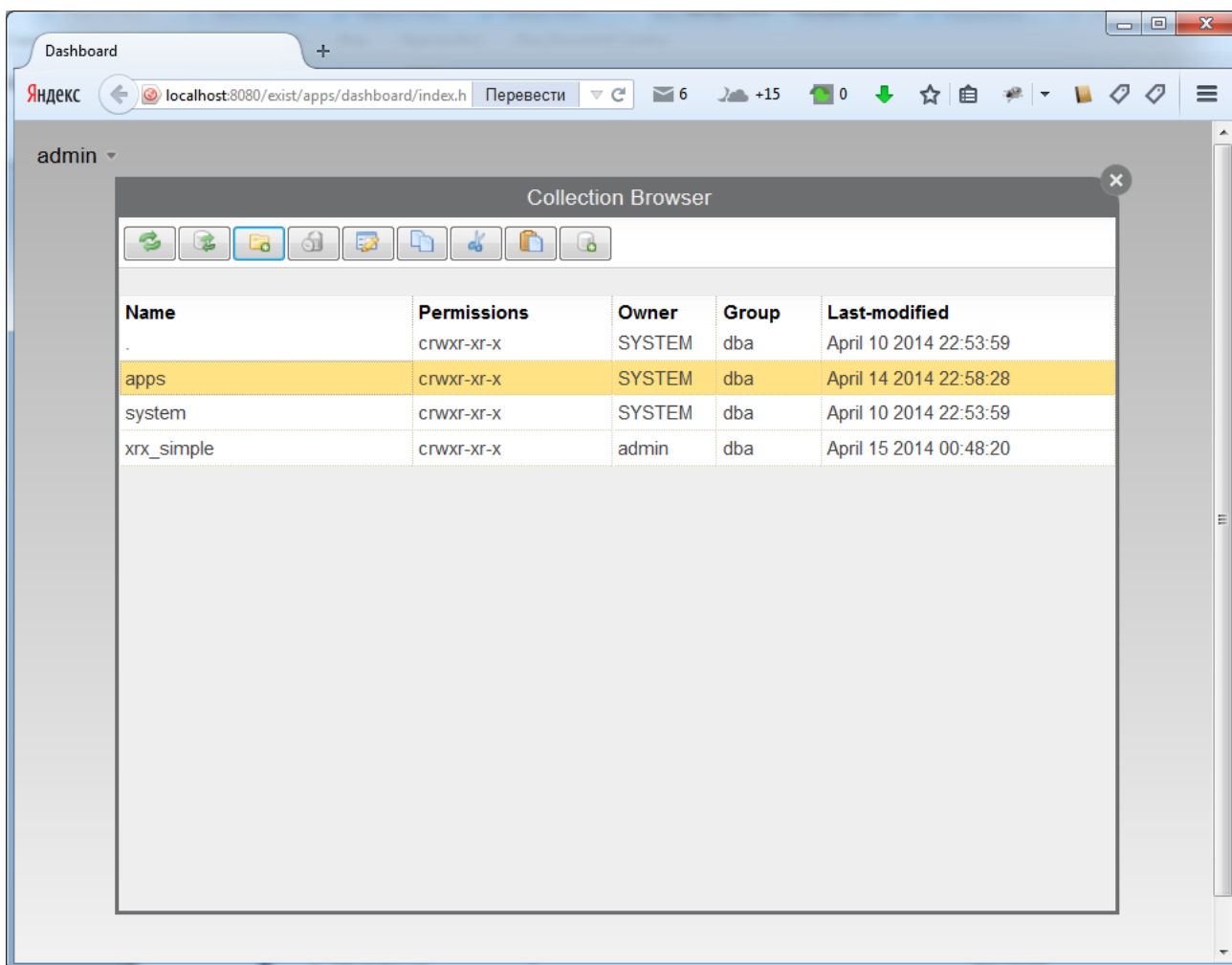


Рис. 4.6. Окно списка коллекций.

Набор кнопок в верхней части окна позволяет выполнять стандартные действия с коллекциями – создание, удаление, обновление списка и др. При наведении на кнопку появляется всплывающая подсказка.

Самая правая кнопка в списке – кнопка загрузки файлов в коллекцию. При нажатии кнопки появляется диалоговое окно загрузки файлов.

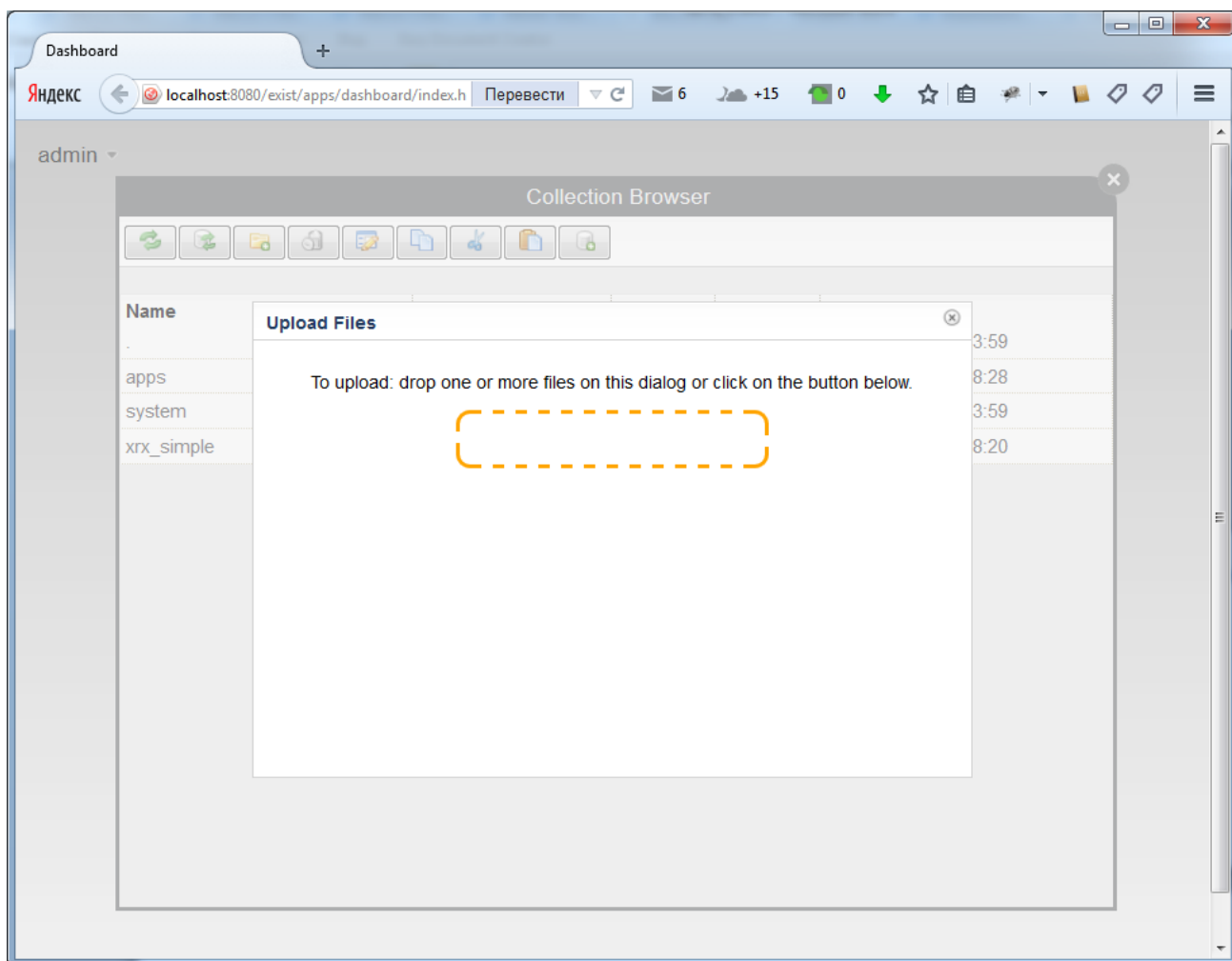


Рис. 4.7. Окно загрузки файлов в коллекцию.

#### 4.2.6 Поддержка WebDAV

Одной из особенностей eXist является возможность работы с ним по протоколу WebDAV.

Протокол WebDAV (Web Distributed Authoring and Versioning) позволяет работать с удаленным веб-ресурсом как с логическим диском в файловой системе.

Данные в eXist хранятся в бинарном виде. Коллекции и файлы XML хранятся не в файловой системе напрямую, а в бинарных файлах. Но к ним можно получить доступ через WebDAV и работать с ними, как с файловой системой.

Для этого нужно добавить новый элемент в сетевое окружение, а в качестве адреса указать «<http://localhost:8080/exist/webdav/db/>». Пользователь: admin, поле



пароля содержит пароль администратора. Результат подключения представлен на следующем рисунке:

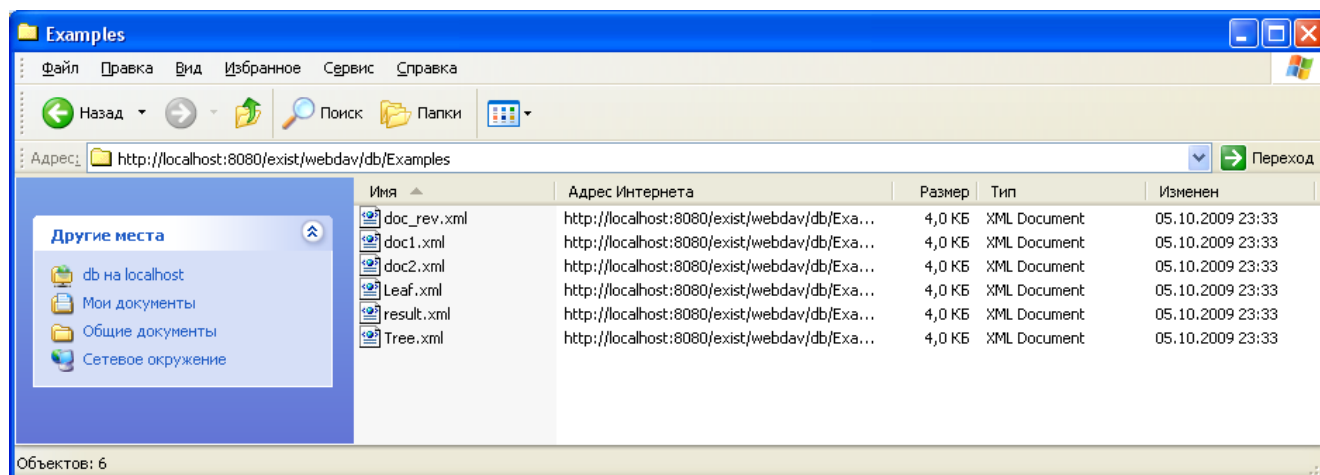


Рис. 4.8. Доступ к eXist с помощью WebDAV.

Работа с WebDAV немного различается в различных версиях eXist. Детальная информация о подключении описана в разделе «WebDAV» справочной системы eXist.

### 4.3 Язык XQuery 1.0

#### 4.3.1 Подготовка и запуск примеров

Для правильной работы XQuery-запросов, которые далее рассматриваются в этом разделе пособия, необходимо создать коллекцию «Examples» в корневой коллекции «db» и загрузить в нее учебные XML файлы из примера 4.1.

Дальнейшие примеры в этом разделе даны в виде таблиц. Левая колонка таблицы содержит текст XQuery-запроса, а правая содержит результат выполнения запроса.

Для запуска примеров используется eXide. Необходимо скопировать текст XQuery-запроса в окно eXide и нажать кнопку «Run» для запуска примера.

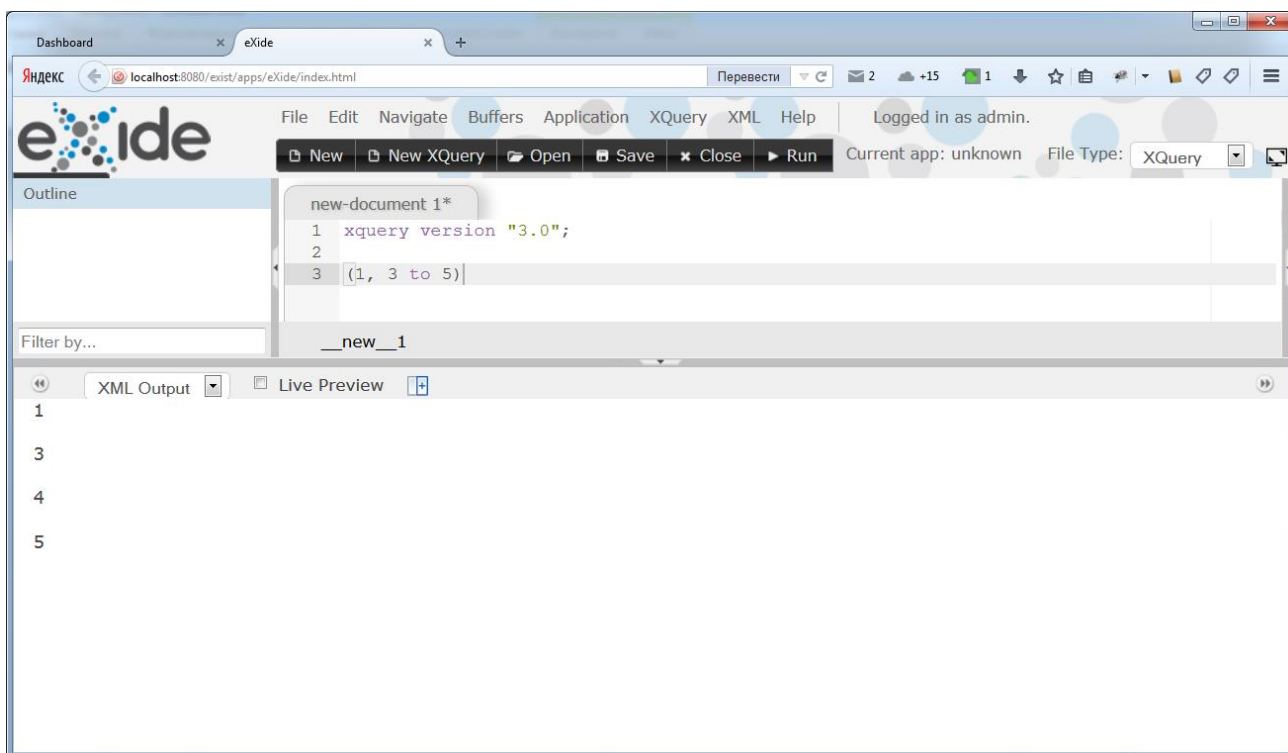


Рис. 4.9. Запуск примера с использованием eXide.

### 4.3.2 Начальные сведения об XQuery

XQuery является чувствительным к регистру символов. Ключевые слова записываются в нижнем регистре. Основным элементом запроса на языке XQuery является выражение (expression), которое обычно записывается в Unicode.

Для обозначения комментариев используются выражения (: и :).

(: Это комментарий :)

Языки XQuery 1.0 и XPath 2.0 почти совпадают. Основным отличием является то, что в XPath 2.0 используется не полный оператор FLWOR (For Let Where OrderBy Return), а сокращенное выражение FOR.

Типы данных, которые используются в языке XQuery, базируются на схемах XML.

Интересным фактом является то, что номинально XQuery относится к классу функциональных языков, таких как Haskell, Erlang, F#, однако он намного проще в освоении, чем другие функциональные языки.

### 4.3.3 Модель данных

Основным элементом данных в языке XQuery является последовательность (sequence). Последовательность – это ноль или более элементов (item).

Элемент является атомарным значением (atomic value) или узлом (node).

Атомарное значение – это значение атомарного типа в соответствии с типами данных XML-схем. Узлом является составной элемент.

Несмотря на то, что у документа XML древовидная структура, основным элементом для XQuery является последовательность. То есть из этой древовидной структуры выбираются последовательности, которые обрабатываются запросами на XQuery.

Более подробно модель данных описана в документе [XDM, 2007].

### 4.3.4 Выражения XPath 1.0

Язык XQuery позволяет использовать выражения XPath 1.0, в том числе выражения с фильтрами.

### 4.3.5 Последовательности

Можно использовать запятую для перечисления элементов или to для задания диапазонов.

Вложенные последовательности не используются. Например, последовательность 1, (2,3) превращается в 1,2,3.

Таблица 4.1. Примеры запросов. Последовательности.

(1, 2, 3)	<b>1, 2, 3</b>
(1, (2, 3), (3, 4), (), (5))	<b>1, 2, 3, 3, 4, 5</b>
(1, 3 to 5)	<b>1, 3, 4, 5</b>
(5 to 3)	<b>пустая последовательность</b>

<code>fn:reverse(3 to 5)</code>	<b>5, 4, 3</b>
<code>'A', 'B'</code>	<b>A, B</b>
<code>"A", "B"</code>	<b>A, B</b>

Возможно использование фильтров:

Таблица 4.2. Примеры запросов. Фильтры.

<code>(1 to 100)[. mod 5 eq 0]</code>	<b>Числа кратные 5 от 5 до 100</b>
<code>(0 to 100)[fn:position() = 1 to 3]</code>	<b>0, 1, 2</b>

#### 4.3.6 Арифметические выражения

- + (сложение)
- - (вычитание)
- \* (умножение)
- div (деление)
- idiv (целая часть от деления)
- mod (остаток от деления)

Таблица 4.3. Примеры запросов. Арифметические выражения.

<code>2+2</code>	<b>4</b>
<code>-(2+2)</code>	<b>-4</b>
<code>3-2</code>	<b>1</b>
<code>5 div 2</code>	<b>2.5</b>
<code>5 idiv 2</code>	<b>2 (целая часть от деления)</b>

<code>5 mod 2</code>	<b>1 (остаток от деления)</b>
----------------------	-------------------------------

### 4.3.7 Операторы сравнения

Таблица 4.4. Операторы сравнения.

Сравнение значений	eq	ne	lt	le	gt	ge
Сравнение последовательностей	=	!=	<	<=	>	>=

Сравнение значений можно выполнять только для атомарных значений (не для последовательностей).

Таблица 4.5. Примеры запросов. Сравнение значений.

<code>2 eq 2</code>	<b>true</b>
<code>3 ne 2</code>	<b>true</b>
<code>(1,2) eq (1,2)</code>	<b>ошибка</b>

Сравнение последовательностей предназначено для одновременного сравнения нескольких элементов последовательностей.

Результат сравнения истинный, если условие выполняется хотя бы для одного любого элемента в первой последовательности и любого элемента во второй последовательности (элементы не обязательно находятся в одинаковой позиции).

Таблица 4.6. Примеры запросов. Сравнение последовательностей.

<code>2 = 2</code>	<b>true</b>
<code>3 != 2</code>	<b>true</b>
<code>(1,2) = (1,3)</code>	<b>true (совпадение для 1)</b>
<code>(1,2) = (3,1)</code>	<b>true (совпадение для 1)</b>

<code>(1,2) = (3,4)</code>	<b>false</b>
----------------------------	--------------

### 4.3.8 Создание элементов (конструкторы элементов)

#### 1. Прямое создание элементов (direct constructors):

Таблица 4.7. Примеры запросов. Прямое создание элементов.

<code>&lt;P&gt;Текст&lt;/P&gt;</code>	<code>&lt;P&gt;Текст&lt;/P&gt;</code>
<code>&lt;DIV&gt;</code> <code>&lt;P&gt;Текст1&lt;/P&gt;</code> <code>&lt;P&gt;Текст2&lt;/P&gt;</code> <code>&lt;/DIV&gt;</code>	<code>&lt;DIV&gt;</code> <code>&lt;P&gt;Текст1&lt;/P&gt;</code> <code>&lt;P&gt;Текст2&lt;/P&gt;</code> <code>&lt;/DIV&gt;</code>
<code>&lt;q xmlns:try="try"&gt;</code> <code>&lt;e id1="1"&gt;1&lt;/e&gt;</code> <code>&lt;e id1="{ (1 to 3) }"&gt;{ (1 to 3) }&lt;/e&gt;</code> <code>&lt;/q&gt;</code>	<code>&lt;q xmlns:try="try"&gt;</code> <code>&lt;e id1="1"&gt;1&lt;/e&gt;</code> <code>&lt;e id1="1 2 3"&gt;1 2 3&lt;/e&gt;</code> <code>&lt;/q&gt;</code>
<code>let \$q:= &lt;q xmlns:try="try"&gt;</code> <code>&lt;e id1="1"&gt;1&lt;/e&gt;</code> <code>&lt;e id1="{ (1 to 3) }"&gt;{ (1 to 3) }&lt;/e&gt;</code> <code>&lt;/q&gt;</code> <code>return \$q</code>	<code>&lt;q xmlns:try="try"&gt;</code> <code>&lt;e id1="1"&gt;1&lt;/e&gt;</code> <code>&lt;e id1="1 2 3"&gt;1 2 3&lt;/e&gt;</code> <code>&lt;/q&gt;</code>

#### 2. Динамическое создание элементов (computed constructors)

Используются следующие конструкции XQuery:

<code>element</code>	название элемента	{значение элемента}
<code>attribute</code>	название атрибута	{значение атрибута}

И название, и значение элементов и атрибутов могут быть созданы динамически.

Таблица 4.8. Примеры запросов. Динамическое создание элементов.

<pre> element a {   attribute b {1},   element c {4},   element d {"value"} } </pre>	<pre> &lt;a b="1"&gt;   &lt;c&gt;4&lt;/c&gt;   &lt;d&gt;value&lt;/d&gt; &lt;/a&gt; </pre>
<pre> let \$q:= 2+3 return element a {   attribute b {1},   element c {\$q},   element d {"value"} } </pre>	<pre> &lt;a b="1"&gt;   &lt;c&gt;5&lt;/c&gt;   &lt;d&gt;value&lt;/d&gt; &lt;/a&gt; </pre>
<pre> for \$i in (1 to 3) let \$n := fn:concat("element",\$i) return element {\$n} {\$i} </pre>	<pre> &lt;element1&gt;1&lt;/element1&gt; &lt;element2&gt;2&lt;/element2&gt; &lt;element3&gt;3&lt;/element3&gt; </pre>

Переименование элемента:  $\$q/@^*$  – копирование атрибутов,  $\$q/^*$  – копирование вложенных элементов.

Таблица 4.9. Примеры запросов. Переименование элемента.

<pre> let \$q:= &lt;q id1="1" id2="2"&gt;&lt;e1&gt;&lt;e2/&gt;&lt;/e1&gt;&lt;/q&gt; return element newname {\$q/@*, \$q/^*} </pre>	<pre> &lt;newname id1="1" id2="2"&gt;   &lt;e1&gt;     &lt;e2/&gt;   &lt;/e1&gt; &lt;/newname&gt; </pre>
--	--

Также возможно использование следующих конструкций:

document {содержимое} – динамическое создание документа

text {“текст”} – создание текста

comment {“комментарий”} – создание комментария

processing-instruction название инструкции {значение инструкции} –  
создание инструкции обработки

### 4.3.9 Выражение FLWOR

Читается как «flower» (цветок). FLWOR – For, Let, Where, Order by, Return.

FOR – создание связанных переменных как переменных цикла

LET – создание одиночных связанных переменных

WHERE – условие

ORDER BY – порядок сортировки

RETURN – возвращаемое значение

Данная конструкция является, пожалуй, наиболее существенным отличием XQuery 1.0 от XPath 2.0. В XPath 2.0 данная конструкция используется в усеченном виде и называется «for»:

FOR – создание связанных переменных как переменных цикла

RETURN – возвращаемое значение

Таблица 4.10. Примеры запросов. Простой пример выражения FLWOR.

<pre> for \$i in (1 to 3) let \$i2:= \$i + 1 return &lt;number&gt; &lt;i&gt;{\$i}&lt;/i&gt; &lt;i2&gt;{\$i2}&lt;/i2&gt; </pre>	<pre> &lt;number&gt;   &lt;i&gt;1&lt;/i&gt; &lt;i2&gt;2&lt;/i2&gt; &lt;/number&gt; &lt;number&gt;   &lt;i&gt;2&lt;/i&gt; &lt;i2&gt;3&lt;/i2&gt; &lt;/number&gt; &lt;number&gt;   &lt;i&gt;3&lt;/i&gt; &lt;i2&gt;4&lt;/i2&gt; &lt;/number&gt; </pre>
--	---



<code>&lt;/number&gt;</code>	
------------------------------	--

Выражения FOR и LET генерируют упорядоченную последовательность кортежей связанных переменных (tuples of bound variables), которая называется потоком кортежей (tuple stream).

Поток кортежей фильтруется в соответствии с условием WHERE.

Поток кортежей сортируется в соответствии с ORDER BY.

Выражение RETURN выполняется для каждого кортежа в потоке кортежей.

Таблица 4.11. Примеры запросов. Примеры на выражение FLWOR.

<pre>for \$i in (1 to 5) let \$i2:= \$i * 2, \$i3:= \$i * 3 return &lt;number i1="{ \$i }" i2="{ \$i2 }" i3="{ \$i3 }" /&gt;</pre>	<pre>&lt;number i1="1" i2="2" i3="3"/&gt; &lt;number i1="2" i2="4" i3="6"/&gt; &lt;number i1="3" i2="6" i3="9"/&gt; &lt;number i1="4" i2="8" i3="12"/&gt; &lt;number i1="5" i2="10" i3="15"/&gt;</pre>	Пример потока кортежей
<pre>for \$i in (1 to 5) let \$i2:= \$i * 2, \$i3:= \$i * 3 where \$i mod 2 != 0 return &lt;number i1="{ \$i }" i2="{ \$i2 }" i3="{ \$i3 }" /&gt;</pre>	<pre>&lt;number i1="1" i2="2" i3="3"/&gt; &lt;number i1="3" i2="6" i3="9"/&gt; &lt;number i1="5" i2="10" i3="15"/&gt;</pre>	Условие выбирает только нечетные элементы

<pre> for \$i in (1 to 5) let \$i2:= \$i * 2, \$i3:= \$i * 3 where \$i mod 2 != 0 order by \$i descending return &lt;number i1="{ \$i}" i2="{ \$i2}" i3="{ \$i3}" /&gt; </pre>	<pre> &lt;number i1="5" i2="10" i3="15"/&gt; &lt;number i1="3" i2="6" i3="9"/&gt; &lt;number i1="1" i2="2" i3="3"/&gt; </pre>	Сортировка в порядке убывания
<pre> let \$doc := &lt;elements&gt; &lt;e&gt;1&lt;/e&gt; &lt;e&gt;2&lt;/e&gt; &lt;e&gt;3&lt;/e&gt; &lt;/elements&gt; for \$i in \$doc/e let \$i2:= \$i * 2, \$i3:= \$i * 3 return &lt;number i1="{ \$i}" i2="{ \$i2}" i3="{ \$i3}" /&gt; </pre>	<pre> &lt;number i1="1" i2="2" i3="3"/&gt; &lt;number i1="2" i2="4" i3="6"/&gt; &lt;number i1="3" i2="6" i3="9"/&gt; </pre>	Перебор элементов документа

Дополнительные примеры:

Таблица 4.12. Примеры запросов. Дополнительные примеры на выражение FLWOR.

<pre> let \$i:= (1 to 3) return &lt;rez&gt;{ \$i}&lt;/rez&gt; </pre>	<pre> &lt;rez&gt;1 2 3&lt;/rez&gt; </pre>	Различие между операторами FOR и
--	---	-------------------------------------

<pre>for \$i in (1 to 3) return &lt;rez&gt;{\$i}&lt;/rez&gt;</pre>	<pre>&lt;rez&gt;1&lt;/rez&gt; &lt;rez&gt;2&lt;/rez&gt; &lt;rez&gt;3&lt;/rez&gt;</pre>	LET
<pre>for \$i in (1 to 3), \$j in (1 to 3) return &lt;rez i="{ \$i}" j="{ \$j}" /&gt;</pre>	<pre>&lt;rez i="1" j="1"/&gt; &lt;rez i="1" j="2"/&gt; &lt;rez i="1" j="3"/&gt; &lt;rez i="2" j="1"/&gt; &lt;rez i="2" j="2"/&gt; &lt;rez i="2" j="3"/&gt; &lt;rez i="3" j="1"/&gt; &lt;rez i="3" j="2"/&gt; &lt;rez i="3" j="3"/&gt;</pre>	Использование нескольких переменных в FOR. Декартово произведение.
<pre>let \$i as xs:decimal :=1 return &lt;rez i="{ \$i}" /&gt;</pre>	<pre>&lt;rez i="1"/&gt;</pre>	Возможно задание типа переменной с помощью конструкции «as тип»

Возможно задание в FOR позиционной переменной с помощью ключевого слова «at». Позиционная переменная всегда типа xs:integer. Она указывает позицию, в которой находится элемент.

Таблица 4.13. Примеры запросов. Использование «at».

<pre>for \$i at \$pos in ("A", "B", "C") return &lt;rez i="{ \$i}" pos="{ \$pos}" /&gt;</pre>	<pre>&lt;rez i="A" pos="1"/&gt; &lt;rez i="B" pos="2"/&gt; &lt;rez i="C" pos="3"/&gt;</pre>	Позиционная переменная
---	---	------------------------

Выражений FOR и LET может быть несколько в одном запросе. Они могут следовать в произвольном порядке.

При совместном использовании нескольких FOR и LET генерируется поток кортежей для переменных всех уровней.

Таблица 4.14. Примеры запросов. Совместное использование FOR и LET.

<pre>let \$i1:=1, \$i2:=3 for \$i in (\$i1 to \$i2) let \$i3 := \$i*2 return &lt;rez i="{ \$i}" i3="{ \$i3}" i1="{ \$i1}" i2="{ \$i2}"/&gt;</pre>	<pre>&lt;rez i="1" i3="2" i1="1" i2="3"/&gt; &lt;rez i="2" i3="4" i1="1" i2="3"/&gt; &lt;rez i="3" i3="6" i1="1" i2="3"/&gt;</pre>	Использование нескольких FOR и LET в одном запросе
<pre>for \$j in 1 to 3 let \$i1:=1, \$i2:=\$j for \$i in (\$i1 to \$i2) let \$i3 := \$i*2 return &lt;rez j="{ \$j}" i="{ \$i}" i3="{ \$i3}" i1="{ \$i1}" i2="{ \$i2}"/&gt;</pre>	<pre>&lt;rez j="1" i="1" i3="2" i1="1" i2="1"/&gt; &lt;rez j="2" i="1" i3="2" i1="1" i2="2"/&gt; &lt;rez j="2" i="2" i3="4" i1="1" i2="2"/&gt; &lt;rez j="3" i="1" i3="2" i1="1" i2="3"/&gt; &lt;rez j="3" i="2" i3="4" i1="1" i2="3"/&gt; &lt;rez j="3" i="3" i3="6" i1="1" i2="3"/&gt;</pre>	
<pre>for \$i in 1 to 3 let \$i:=\$i*2 return &lt;rez i="{ \$i}"/&gt;</pre>	<pre>&lt;rez i="2"/&gt; &lt;rez i="4"/&gt; &lt;rez i="6"/&gt;</pre>	Возможно переопределение имени переменной

Значение выражения WHERE вычисляется для каждого кортежа в потоке кортежей. Если значение = false, то кортеж удаляется.

В случае использования нескольких переменных в FOR условие WHERE может быть использовано для связи между ними (join).

Таблица 4.15. Примеры запросов. Использование WHERE.

<pre>for \$i in (1 to 3), \$j in (1 to 3) where \$i=\$j and \$i &gt; 1 return &lt;rez i="{ \$i}" j="{ \$j}" /&gt;</pre>	<pre>&lt;rez i="2" j="2"/&gt; &lt;rez i="3" j="3"/&gt;</pre>	<p>Для чисел</p>
<pre>let \$doc1 := &lt;elements&gt;&lt;e id="1"&gt;1&lt;/e&gt;&lt;e id="2"&gt;2&lt;/e&gt;&lt;e id="3"&gt;3&lt;/e&gt;&lt;/elements&gt;,   \$doc2 := &lt;values&gt;&lt;v id1="1" value="value1"/&gt; &lt;v id1="2" value="value2"/&gt; &lt;/values&gt; for \$i in \$doc1/e, \$j in \$doc2/v return &lt;rez&gt;{ \$i } { \$j } &lt;/rez&gt;</pre>	<pre>&lt;rez&gt; &lt;e id="1"&gt;1&lt;/e&gt; &lt;v id1="1" value="value1"/&gt; &lt;/rez&gt; &lt;rez&gt; &lt;e id="1"&gt;1&lt;/e&gt; &lt;v id1="2" value="value2"/&gt; &lt;/rez&gt; &lt;rez&gt; &lt;e id="2"&gt;2&lt;/e&gt; &lt;v id1="1" value="value1"/&gt; &lt;/rez&gt; &lt;rez&gt; &lt;e id="2"&gt;2&lt;/e&gt; &lt;v id1="2" value="value2"/&gt; &lt;/rez&gt; &lt;rez&gt; &lt;e id="3"&gt;3&lt;/e&gt; &lt;v id1="1" value="value1"/&gt; &lt;/rez&gt; &lt;rez&gt; &lt;e id="3"&gt;3&lt;/e&gt; &lt;v id1="2" value="value2"/&gt; &lt;/rez&gt;</pre>	<p>Для элементов. Без фильтра.</p>

<pre> let \$doc1 := &lt;elements&gt;&lt;e id="1"&gt;1&lt;/e&gt;&lt;e id="2"&gt;2&lt;/e&gt;&lt;e id="3"&gt;3&lt;/e&gt;&lt;/elements&gt;,    \$doc2 := &lt;values&gt;&lt;v id1="1" value="value1"/&gt; &lt;v id1="2" value="value2"/&gt; &lt;/values&gt;  for \$i in \$doc1/e, \$j in \$doc2/v    where \$i/@id=\$j/@id1    return  &lt;rez&gt;{\$i}{\$j}&lt;/rez&gt; </pre>	<pre> &lt;rez&gt; &lt;e id="1"&gt;1&lt;/e&gt; &lt;v id1="1" value="value1"/&gt; &lt;/rez&gt; &lt;rez&gt; &lt;e id="2"&gt;2&lt;/e&gt; &lt;v id1="2" value="value2"/&gt; &lt;/rez&gt; </pre>	<p>Для элементов. С фильтром.</p>
--	--	---

Сортировка с использованием ORDER BY:

Таблица 4.16. Примеры запросов. Использование ORDER BY.

<pre> let \$doc := &lt;elements&gt;&lt;e&gt;1&lt;/e&gt;&lt;e&gt;2&lt;/e&gt; &lt;e/&gt;&lt;/elements&gt;  for \$i in \$doc/e  order by \$i ascending  return \$i </pre>	<pre> &lt;e/&gt; &lt;e&gt;1&lt;/e&gt; &lt;e&gt;2&lt;/e&gt; </pre>	<p>Сортировка по возрастанию</p>
<pre> let \$doc := &lt;elements&gt;&lt;e&gt;1&lt;/e&gt;&lt;e&gt;2&lt;/e&gt; &lt;e/&gt; &lt;/elements&gt;  for \$i in \$doc/e  order by \$i descending  return \$i </pre>	<pre> &lt;e&gt;2&lt;/e&gt; &lt;e&gt;1&lt;/e&gt; &lt;e/&gt; </pre>	<p>Сортировка по убыванию</p>

<pre> let \$doc := &lt;elements&gt;&lt;e&gt;1&lt;/e&gt;&lt;e&gt;2&lt;/e&gt; &lt;e/&gt;&lt;/elements&gt;  for \$i in \$doc/e order by \$i ascending empty greatest return \$i </pre>	<pre> &lt;e/&gt; &lt;e&gt;1&lt;/e&gt; &lt;e&gt;2&lt;/e&gt; </pre> <p>Ожидаемый результат:</p> <pre> &lt;e&gt;1&lt;/e&gt; &lt;e&gt;2&lt;/e&gt; &lt;e/&gt; </pre>	<p>Пример empty.</p>
---	---	--------------------------

Выражения empty greatest (пустые значения считаются большими) и empty least (считаются меньшими) вероятнее всего не работают в текущей версии eXist.

#### 4.3.10 Соединение документов с помощью выражения FLWOR

Задача связи (join) между двумя документами может быть решена тремя способами.

- с помощью for
- с помощью let
- с помощью where

В примере используются файлы «doc1.xml» и «doc2.xml» следующей структуры:

```

<result>
  <rez>
    <doc>
      <e>1</e>
      <e>1</e>
      ...
      <e>500</e>
    </doc>
  </rez>
</result>

```

В файле «doc\_rev.xml» элементы расположены в обратном порядке.

Для заполнения документов данными необходимо выполнить следующие запросы. В запросах используются операции обновления данных, которые рассматриваются далее в этой главе.

Заполнение документа «doc1.xml»:

```
let
$doc := <doc>{
for $i in (1 to 500)
return <e>{$i}</e>
}</doc>,
$file := doc("/db/Examples/doc1.xml")
return
update insert $doc into $file//rez
```

Заполнение документа «doc2.xml»:

```
let
$doc := <doc>{
for $i in (1 to 500)
return <e>{$i}</e>
}</doc>,
$file := doc("/db/Examples/doc2.xml")
return
update insert $doc into $file//rez
```



Заполнение документа «doc\_rev.xml»:

```
let $doc := <doc>{
for $i in fn:reverse(1 to 500)
return <e>{$i}</e>
}</doc>,
$file := doc("/db/Examples/doc_rev.xml")
return
update insert $doc into $file//rez
```

Примеры соединения:

Таблица 4.17. Примеры запросов. Соединение документов.

<pre>let \$doc1 := doc("/db/Examples/doc1.xml"), \$doc2 := doc("/db/Examples/doc2.xml") for \$i in \$doc1//e, \$j in \$doc2//e where \$i=\$j return &lt;p&gt;{\$i}{\$j}&lt;/p&gt;</pre>	<p>Связывание с помощью where.</p> <p>Время выполнения около 1.5 сек.</p> <p>Если вместо doc2.xml использовать файл doc_rev.xml, то время выполнения почти не изменяется.</p>
<pre>let \$doc1 := doc("/db/Examples/doc1.xml"), \$doc2 := doc("/db/Examples/doc2.xml") for \$i in \$doc1//e let \$j:=\$doc2//e[.= \$i] return &lt;p&gt;{\$i}{\$j}&lt;/p&gt;</pre>	<p>Связывание с помощью let.</p> <p>Время выполнения около 1 сек.</p> <p>Если вместо doc2.xml использовать файл doc_rev.xml, то время выполнения почти не изменяется.</p>

<pre> let \$doc1 := doc("/db/Examples/doc1.xml") , \$doc2 := doc("/db/Examples/doc2.xml") for \$i in \$doc1//e, \$j in \$doc2//e[.=\$i] return &lt;p&gt;{\$i}{\$j}&lt;/p&gt; </pre>	<p>Связывание с помощью for.</p> <p>Время выполнения около 1 сек.</p> <p>Если вместо doc2.xml использовать файл doc_rev.xml, то время выполнения почти не изменяется.</p>
---	---

Если элементы в разных документах расположены в обратном порядке, то на скорость выполнения запроса это почти не влияет.

В данном примере показано, что в eXist соединение с помощью where медленнее, чем с помощью let или for. Но на это может влиять много факторов – наличие индексов, и т.д. В другой XML-ориентированной СУБД, наоборот, соединение с помощью where может быть быстрее.

#### 4.3.11 Условное выражение if-then-else

Таблица 4.18. Примеры запросов. Условное выражение.

<pre> if (1=1) then &lt;rez&gt;1&lt;/rez&gt; else &lt;rez&gt;2&lt;/rez&gt; </pre>	<pre>&lt;rez&gt;1&lt;/rez&gt;</pre>	Выражение может использоваться отдельно или внутри FLWOR
<pre> let \$doc := &lt;elements&gt;&lt;e&gt;1&lt;/e&gt;&lt;e&gt;2&lt;/e&gt; &lt;e&gt;3&lt;/e&gt;&lt;/elements&gt; for \$i in \$doc/e let \$j := ( if (\$i=3) then &lt;e&gt;333&lt;/e&gt; else \$i ) </pre>	<pre> &lt;e&gt;1&lt;/e&gt; &lt;e&gt;2&lt;/e&gt; &lt;e&gt;333&lt;/e&gt; </pre>	Использование if внутри let

<code>return \$j</code>		
<pre>let \$doc := &lt;elements&gt;&lt;e&gt;1&lt;/e&gt;&lt;e&gt;2&lt;/e&gt; &lt;e&gt;3&lt;/e&gt;&lt;/elements&gt; for \$i in \$doc/e return if (\$i=3) then &lt;e&gt;333&lt;/e&gt; else \$i</pre>	<pre>&lt;e&gt;1&lt;/e&gt; &lt;e&gt;2&lt;/e&gt; &lt;e&gt;333&lt;/e&gt;</pre>	Использование if внутри return
<pre>let \$doc := &lt;elements&gt;&lt;e&gt;1&lt;/e&gt;&lt;e&gt;2&lt;/e&gt; &lt;e&gt;3&lt;/e&gt;&lt;/elements&gt; for \$i in \$doc/e return if ((\$i=2 or \$i=3) and fn:not(\$i=1)) then &lt;e&gt;333&lt;/e&gt; else \$i</pre>	<pre>&lt;e&gt;1&lt;/e&gt; &lt;e&gt;333&lt;/e&gt; &lt;e&gt;333&lt;/e&gt;</pre>	Использование or, and и функции fn:not

#### 4.3.12 Выражения some и every

Выражения `some` и `every` в спецификации также называются Quantified Expressions. Синтаксис:

**SOME | EVERY \$переменная IN выражение (, \$переменная\_n IN выражение\_n) \* SATISFIES условие**

Выражение всегда возвращает `true` или `false`. `EVERY` (И) означает, что условие должно выполняться для всех значений, `SOME` (ИЛИ) – хотя бы для одного.

Таблица 4.19. Примеры запросов. Выражения `some` и `every`.

<code>some \$i in (1 to 3) satisfies \$i&lt;2</code>	<b>true</b>	
<code>every \$i in (1 to 3) satisfies \$i&lt;2</code>	<b>false</b>	

<pre> let \$rez := &lt;rez&gt;{ for \$i in (1 to 3) return &lt;r&gt;{\$i&lt;2&gt;&lt;/r&gt; }&lt;/rez&gt;  return    if (fn:count(\$rez/r[. = fn:true()]) &gt; 0 )    then fn:true()    else fn:false() </pre>	<b>true</b>	<p>Аналогичные выражения с использованием FLWOR.</p> <p>Аналог SOME</p>
<pre> let \$rez := &lt;rez&gt;{ for \$i in (1 to 3) return &lt;r&gt;{\$i&lt;2&gt;&lt;/r&gt; }&lt;/rez&gt;  return    if (fn:count(\$rez/r[. = fn:false()]) &gt; 0 )    then fn:false()    else fn:true() </pre>	<b>false</b>	<p>Аналог EVERY</p>
<pre> some \$x in (1, 2, 3), \$y in (2, 3, 4) satisfies \$x + \$y = 4 </pre>	<b>true</b>	<p>Возможно указание нескольких переменных (пример из спецификации)</p>
<pre> some \$x as xs:integer in (1, 2, 3), \$y as xs:integer in (2, 3, 4) satisfies \$x + \$y = 4 </pre>	<b>true</b>	<p>Возможно указание типов данных с помощью AS</p>

<pre> let \$doc1 := doc("/db/Examples/doc1.xml"), \$doc2 := doc("/db/Examples/doc2.xml") return some \$i in \$doc1//e, \$j in \$doc2//e satisfies \$i=\$j </pre>	<b>true</b>	Строится декартово произведение элементов «e» из первого и второго документа,
<pre> let \$doc1 := doc("/db/Examples/doc1.xml"), \$doc2 := doc("/db/Examples/doc2.xml") return every \$i in \$doc1//e, \$j in \$doc2//e satisfies \$i=\$j </pre>	<b>false</b>	поэтому some возвращает true, а every возвращает false
<pre> let \$rez := &lt;q&gt; { let \$doc1 := doc("/db/Examples/doc1.xml"), \$doc2 := doc("/db/Examples/doc2.xml") for \$i in \$doc1//e, \$j in \$doc2//e where \$i=\$j return &lt;p&gt;&lt;i&gt;{\$i}&lt;/i&gt;&lt;j&gt;{\$j}&lt;/j&gt;&lt;/p&gt; } &lt;/q&gt; return every \$i1 in \$rez/p/i, \$j1 in \$rez/p/j satisfies \$i1=\$j1 </pre>	<b>false</b>	В переменной \$rez в элементах <p> i=j. Но выражения \$rez/p/i и \$rez/p/j разделяют выборку на отдельные подзапросы, по которым строится декартово произведение. В декартовом произведении не

		всегда $i=j$ . Поэтому every возвращает false.
<pre> let \$rez := &lt;q&gt; { let \$doc1 := doc("/db/Examples/doc1.xml"), \$doc2 := doc("/db/Examples/doc2.xml") for \$i in \$doc1//e, \$j in \$doc2//e where \$i=\$j return &lt;p&gt;&lt;i&gt;{\$i}&lt;/i&gt;&lt;j&gt;{\$j}&lt;/j&gt;&lt;/p&gt; } &lt;/q&gt; return every \$p in \$rez/p satisfies \$p/i = \$p/j </pre>	true	Правильный вариант

Следующие типы конструкций XQuery, как правило, могут использоваться реже, чем предыдущие.

#### 4.3.13 Действия над множествами

Возможны следующие действия над множествами:

- union (или |) – объединение множеств
- intersect – пересечение множеств
- except – первое множество минус второе множество

Действия нельзя выполнять над атомарными элементами, только над узлами.

При выполнении действий над множествами используется понятие идентичности (identity) элементов. Каждый элемент идентичен себе и не идентичен другим элементам.

При выполнении действий два элемента считаются одинаковыми, если они идентичны (то есть это один и тот же элемент). Если два полностью совпадающих элемента находятся в разных документах (или разных частях документа), то они

не идентичны и не будут считаться совпадающими при выполнении действий над множествами.

Таблица 4.20. Примеры запросов. Действия над множествами.

<pre>let \$set1:= &lt;q&gt;&lt;e&gt;1&lt;/e&gt;&lt;e&gt;2&lt;/e&gt;&lt;e&gt;3&lt;/e&gt;&lt;/q&gt;, \$set2:= &lt;q&gt;&lt;e&gt;1&lt;/e&gt;&lt;e&gt;2&lt;/e&gt;&lt;/q&gt; return \$set1/e union \$set2/e</pre>	<p>Запрос не будет работать правильно, так как одинаковые элементы в двух переменных не идентичны</p>
<pre>let \$set:= &lt;q&gt; &lt;e id1="1" id2="2"&gt;1&lt;/e&gt; &lt;e id1="1" id2="2"&gt;2&lt;/e&gt; &lt;e id1="1"&gt;3&lt;/e&gt; &lt;e id2="2"&gt;4&lt;/e&gt; &lt;/q&gt; return \$set/e[@id1="1"] union \$set/e[@id2="2"]</pre>	<p>Все 4 элемента «e»</p>
<pre>let \$set:= &lt;q&gt; &lt;e id1="1" id2="2"&gt;1&lt;/e&gt; &lt;e id1="1" id2="2"&gt;2&lt;/e&gt; &lt;e id1="1"&gt;3&lt;/e&gt; &lt;e id2="2"&gt;4&lt;/e&gt; &lt;/q&gt; return \$set/e[@id1="1"] intersect \$set/e[@id2="2"]</pre>	<p>Элементы 1 и 2</p>

<pre> let \$set:= &lt;q&gt; &lt;e id1="1" id2="2"&gt;1&lt;/e&gt; &lt;e id1="1" id2="2"&gt;2&lt;/e&gt; &lt;e id1="1"&gt;3&lt;/e&gt; &lt;e id2="2"&gt;4&lt;/e&gt; &lt;/q&gt;  return \$set/e[@id1="1"] except \$set/e[@id2="2"] </pre>	Элемент 3
--	-----------

#### 4.3.14 Сравнение узлов

- «is» возвращает истину, если узлы идентичны (являются одним и тем же узлом).
- «<<» возвращает истину, если узел расположен в документе ранее другого.
- «>>» возвращает истину, если узел расположен в документе позднее другого.

Таблица 4.21. Примеры запросов. Сравнение узлов.

<pre> let \$q:= &lt;q&gt;&lt;e&gt;1&lt;/e&gt;&lt;e&gt;2&lt;/e&gt;&lt;e&gt;1&lt;/e&gt;&lt;/q&gt; return \$q/e[1] = \$q/e[3] </pre>	true	Первый и третий элементы «e» равны, но не являются одним и тем же элементом
<pre> let \$q:= &lt;q&gt;&lt;e&gt;1&lt;/e&gt;&lt;e&gt;2&lt;/e&gt;&lt;e&gt;1&lt;/e&gt;&lt;/q&gt; return \$q/e[1] is \$q/e[3] </pre>	false	
<pre> let \$q:= &lt;q&gt;&lt;e&gt;1&lt;/e&gt;&lt;e&gt;2&lt;/e&gt;&lt;e&gt;1&lt;/e&gt;&lt;/q&gt; return \$q/e[1] &lt;&lt; \$q/e[3] </pre>	true	



<pre>let \$q:= &lt;q&gt;&lt;e&gt;1&lt;/e&gt;&lt;e&gt;2&lt;/e&gt;&lt;e&gt;1&lt;/e&gt;&lt;/q&gt; return \$q/e[1] &gt;&gt; \$q/e[3]</pre>	false	
--	-------	--

#### 4.3.15 Выражения **ordered** и **unordered**

Если выражение находится внутри **unordered** { . . }, это означает, что порядок выборки элементов не является важным и СУБД может выбирать элементы так, чтобы это было наиболее эффективно. Теоретически это может ускорить выполнение запроса. Существует аналогичная функция `fn:unordered()`.

Выражение **ordered** { . . } оказывает обратное действие и может замедлить выполнение запроса.

#### 4.3.16 Выражения для работы с типами данных

Иерархия типов данных в XQuery показана на следующем рисунке:

# XPath 2.0 and XQuery 1.0 Type Hierarchy

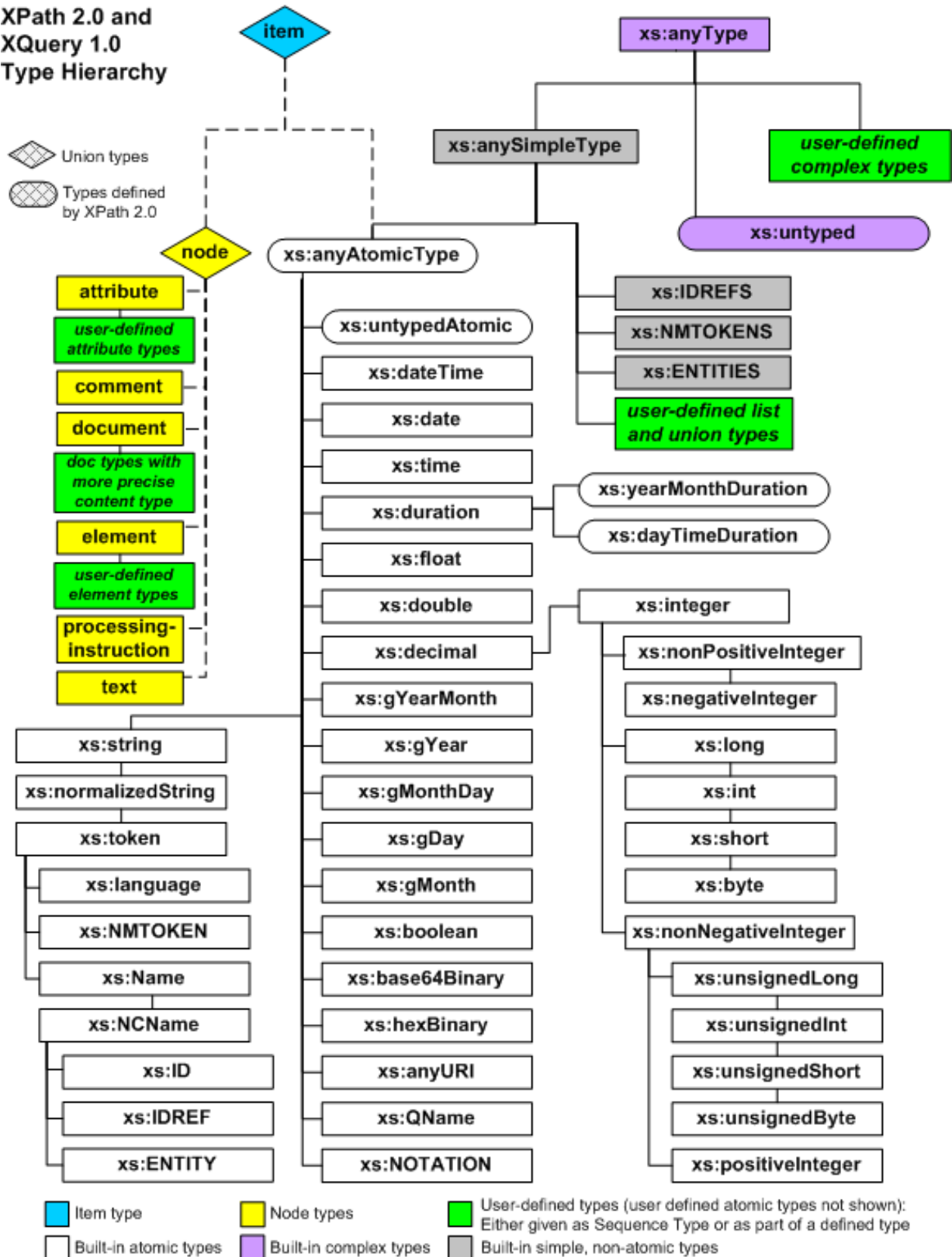


Рис. 4.10. Иерархия типов данных в XQuery.

Рассмотрим различные варианты действий, которые можно осуществлять на основе типов.

Конструкторы типов.

xs:тип(выражение) – приведение выражения к данному типу.

Таблица 4.22. Примеры запросов. Конструкторы типов.

<pre>for \$i in (1.12345678912345, 2.222, 3.333) return &lt;float&gt;{\$i} - {xs:float(\$i)}&lt;/float&gt;</pre>	<pre>&lt;float&gt;1.12345678912345 - 1.1234568&lt;/float&gt; &lt;float&gt;2.222 - 2.222&lt;/float&gt; &lt;float&gt;3.333 - 3.333&lt;/float&gt;</pre>
--	--

Выражение instance of Тип.

Проверка того, что выражение заданного типа.

Таблица 4.23. Примеры запросов. Выражение instance of.

333 instance of xs:integer	true
----------------------------	------

Выражение typeswitch – ветвление по типу.

Таблица 4.24. Примеры запросов. Выражение typeswitch.

<pre>for \$i in (1, xs:float(2.5), "str") return   typeswitch(\$i)     case \$i as xs:integer return   &lt;integer&gt;{\$i+1}&lt;/integer&gt;     case \$i as xs:float return   &lt;float&gt;{\$i*2}&lt;/float&gt;     case \$i as xs:string return   &lt;string&gt;{\$i}&lt;/string&gt;   default return "???"</pre>	<pre>&lt;integer&gt;2&lt;/integer&gt; &lt;float&gt;5&lt;/float&gt; &lt;string&gt;str&lt;/string&gt;</pre>
---	---

Выражения `castable` (проверка возможности приведения к типу данных) и `cast` (приведение к типу данных).

Таблица 4.25. Примеры запросов. Выражения `castable` и `cast`.

<pre>for \$i in (1, xs:float(2.5), "str") return   if (\$i castable as xs:float)   then \$i cast as xs:float   else if (\$i castable as xs:integer)   then \$i cast as xs:integer   else \$i cast as xs:string</pre>	<b>1 2.5 str</b>
--	------------------

#### 4.3.17 Обновление данных в базе данных (XQuery Update Facility)

В таблице приведены лишь некоторые примеры запросов на обновление данных, которые работают в eXist. Полный вариант можно найти в спецификации [XQuery Update, 2011]. Авторы eXist отмечают, что возможности обновления данных появились в СУБД задолго до появления спецификации, поэтому возможности обновления данных в eXist не до конца соответствуют стандарту.

Таблица 4.26. Примеры запросов. Обновление данных.

<pre>let \$b := doc("/db/Examples/Tree.xml")//branch[@id=5] return update insert &lt;insert_into/&gt; into \$b</pre>	Вставка содержимого элемента
<pre>let \$b := doc("/db/Examples/Tree.xml")//branch[@id=5] return update insert &lt;insert_preceding/&gt; preceding \$b</pre>	Вставка до элемента

<pre>let \$b := doc("/db/Examples/Tree.xml")//branch[@id=5] return update insert &lt;insert_following/&gt; following \$b</pre>	Вставка после элемента
<pre>let \$b := doc("/db/Examples/Tree.xml")//branch[@id=5] return update replace \$b with &lt;Replaced/&gt;</pre>	Замена элемента
<pre>let \$b := doc("/db/Examples/Tree.xml")//branch[@id=2] return update value \$b with &lt;Update_Value/&gt;</pre>	Замена содержимого элемента
<pre>let \$b := doc("/db/Examples/Tree.xml")//branch[@id=4] return update rename \$b as &lt;tag&gt;Branch_Renamed&lt;/tag&gt;</pre>	Переименование элемента
<pre>let \$b := doc("/db/Examples/Tree.xml")//branch[@id=2] return update delete \$b</pre>	Удаление

#### 4.4 Расширения XQuery 3.0

В настоящее время разработан стандарт XQuery 3.0 [XQuery, 2014]. Можно предположить, что номер версии 2.0 не был использован для обеспечения совместимости номеров версий с XPath/XSLT.

В стандарте XQuery 3.0 предложены полезные расширения языка, большая часть этих расширений поддерживается в eXist. Эти расширения рассмотрены в следующих примерах.

### 4.4.1 Группировка

В стандарте XQuery 3.0 оператор FLWOR дополнен выражением GROUP BY для группировки элементов. Необходимо отметить, что отсутствие этого оператора не было серьезной проблемой в XQuery 1.0, так как вместо него использовалась функция distinct-values.

Пример реализации группировки средствами XQuery 1.0 и XQuery 3.0 приведен в следующей таблице.

Таблица 4.27. Примеры запросов. Группировка данных.

<pre>xquery version "1.0"; let \$doc := &lt;data&gt; &lt;d dep="отдел кадров" emp="Иванов И.И." /&gt; &lt;d dep="отдел разработки" emp="Петров П.П." /&gt; &lt;d dep="отдел разработки" emp="Сидоров А.И." /&gt; &lt;/data&gt;, \$deps := fn:distinct-values(\$doc//d/@dep) for \$dep in \$deps let \$count := fn:count(\$doc//d[@dep=\$dep]) return &lt;dep name="{ \$dep}" NumberOfEmps="{ \$count}" /&gt;</pre>	<pre>&lt;dep name="отдел кадров" NumberOfEmps="1"/&gt; &lt;dep name="отдел разработки" NumberOfEmps="2"/&gt;</pre>
<pre>xquery version "3.0"; let \$doc := &lt;data&gt; &lt;d dep="отдел кадров" emp="Иванов И.И." /&gt; &lt;d dep="отдел разработки" emp="Петров П.П." /&gt; &lt;d dep="отдел разработки" emp="Сидоров А.И." /&gt; &lt;/data&gt; for \$d in \$doc//d let \$dep := \$d/@dep group by \$dep return &lt;dep name="{ \$dep}" NumberOfEmps="{ fn:count(\$doc//d[@dep=\$dep]) }" /&gt;</pre>	<pre>&lt;dep name="отдел кадров" NumberOfEmps="1"/&gt; &lt;dep name="отдел разработки" NumberOfEmps="2"/&gt;</pre>

### 4.4.1 Оператор ветвления

В версии XQuery 3.0 появился новый оператор ветвления switch-case. Пример использования оператора:

Таблица 4.28. Примеры запросов. Оператор ветвления.

<pre>xquery version "3.0"; let \$lang := 'XML' return switch (\$lang)   case 'XML' return 'Markup Language'   case 'C#' return 'Programming Language'   default return 'Unknown Language'</pre>	<p><b>Markup Language</b></p>
---	-------------------------------

### 4.4.2 Обработка исключений

В версии XQuery 3.0 появился новый оператор try-catch для обработки исключений.

Блок try { ... } содержит выполняемые операторы.

Блок catch \* { ... } содержит обработчик исключений. Символ «\*» означает что блок перехватывает все исключения, вместо «\*» может стоять наименование конкретного исключения.

Пример использования оператора:

Таблица 4.29. Примеры запросов. Обработка исключений.

<pre>xquery version "3.0"; try { let \$q := 1 / 0 return \$q } catch * {   &lt;error&gt;   &lt;code&gt;{\$err:code}&lt;/code&gt;   &lt;description&gt;{\$err:description}&lt;/description&gt;   &lt;/error&gt; }</pre>	<pre>&lt;error&gt; &lt;code&gt;err:XPTY0019&lt;/code&gt; &lt;description&gt;It is a type error if the result of a step (other than the last step) in a path expression contains an atomic value.&lt;/description&gt; &lt;/error&gt;</pre>
--	---

### 4.4.3 Оператор map (!)

Язык XQuery относится к классу функциональных языков. В версии 3.0 в него добавлен оператор мэппинга, который присутствует в большинстве функциональных языков. Оператор мэппинга похож на цикл for или на программный конвейер. Этот оператор позволяет выполнять одинаковое действие для множества операндов. Примеры использования оператора приведены в таблице:

Таблица 4.30. Примеры запросов. Оператор мэппинга.

<pre>xquery version "3.0"; (0,1,2) ! (. + 10)</pre>	<p><b>10 11 12</b></p>
<pre>xquery version "3.0"; ('XML', 'C#', 'HTML') ! &lt;data&gt;   &lt;str&gt;{.}&lt;/str&gt;   &lt;len&gt;{string-length(.)}&lt;/len&gt; &lt;/data&gt; ! &lt;res&gt;Длина строки '{xs:string(./str)}' составляет {xs:integer(./len)}.&lt;/res&gt;</pre>	<p><b>&lt;res&gt;Длина строки 'XML'</b> <b>составляет 3.&lt;/res&gt;</b></p> <p><b>&lt;res&gt;Длина строки 'C#'</b> <b>составляет 2.&lt;/res&gt;</b></p> <p><b>&lt;res&gt;Длина строки 'HTML'</b> <b>составляет 4.&lt;/res&gt;</b></p>

## 4.5 Технология XQueryX

В отличие от XSLT язык XQuery не использует теговый синтаксис XML. Для решения данной задачи был разработан стандарт XQueryX [XQueryX, 2014], который позволяет сохранить XQuery-запрос в виде XML-документа.

Данный формат является очень некомпактным и неудобным для чтения человеком, поэтому его лучше рассматривать как способ сериализации-десериализации XQuery-документов.



## Пример запроса на языке XQuery (из спецификации XQueryX):

```
<bib>
{
  for $b in doc("http://bstore1.example.com/bib.xml")/bib/book
  where $b/publisher = "Addison-Wesley" and $b/@year > 1991
  return
    <book year="{ $b/@year }">
      { $b/title }
    </book>
}
</bib>
```

## Представление запроса в формате XQueryX:

```
<?xml version="1.0"?>
<xqx:module xmlns:xqx="http://www.w3.org/2005/XQueryX"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2005/XQueryX
    http://www.w3.org/2005/XQueryX/xqueryx.xsd">

  <xqx:mainModule>
    <xqx:queryBody>
      <xqx:elementConstructor>
        <xqx:tagName>bib</xqx:tagName>
        <xqx:elementContent>
          <xqx:flworExpr>
            <xqx:forClause>
              <xqx:forClauseItem>
                <xqx:typedVariableBinding>
                  <xqx:varName>b</xqx:varName>
                </xqx:typedVariableBinding>
                <xqx:forExpr>
                  <xqx:pathExpr>
                    <xqx:stepExpr>
                      <xqx:filterExpr>
                        <xqx:functionCallExpr>
                          <xqx:functionName>doc</xqx:functionName>
                          <xqx:arguments>
                            <xqx:stringConstantExpr>
                              <xqx:value>http://bstore1.example.com/bib.xml</xqx:value>
                            </xqx:stringConstantExpr>
                          </xqx:arguments>
                        </xqx:functionCallExpr>
                      </xqx:filterExpr>
                    </xqx:stepExpr>
                  <xqx:stepExpr>
                    <xqx:xpathAxis>child</xqx:xpathAxis>
                    <xqx:nameTest>bib</xqx:nameTest>
                  </xqx:stepExpr>
                <xqx:stepExpr>
                  <xqx:xpathAxis>child</xqx:xpathAxis>
                  <xqx:nameTest>book</xqx:nameTest>
                </xqx:stepExpr>
              </xqx:pathExpr>
            </xqx:forExpr>
          </xqx:forClauseItem>
        </xqx:forClause>
        <xqx:whereClause>
          <xqx:andOp>
            <xqx:firstOperand>
              <xqx:equalOp>
                <xqx:firstOperand>
                  <xqx:pathExpr>
                    <xqx:stepExpr>
                      <xqx:filterExpr>
                        <xqx:varRef>
                          <xqx:name>b</xqx:name>
```

[Оглавление](#)

```

        </xqx:varRef>
      </xqx:filterExpr>
    </xqx:stepExpr>
  <xqx:stepExpr>
    <xqx:xpathAxis>child</xqx:xpathAxis>
    <xqx:nameTest>publisher</xqx:nameTest>
  </xqx:stepExpr>
</xqx:pathExpr>
</xqx:firstOperand>
<xqx:secondOperand>
  <xqx:stringConstantExpr>
    <xqx:value>Addison-Wesley</xqx:value>
  </xqx:stringConstantExpr>
</xqx:secondOperand>
</xqx:equalOp>
</xqx:firstOperand>
<xqx:secondOperand>
  <xqx:greaterThanOp>
    <xqx:firstOperand>
      <xqx:pathExpr>
        <xqx:stepExpr>
          <xqx:filterExpr>
            <xqx:varRef>
              <xqx:name>b</xqx:name>
            </xqx:varRef>
          </xqx:filterExpr>
        </xqx:stepExpr>
      <xqx:stepExpr>
        <xqx:xpathAxis>attribute</xqx:xpathAxis>
        <xqx:nameTest>year</xqx:nameTest>
      </xqx:stepExpr>
    </xqx:pathExpr>
  </xqx:firstOperand>
  <xqx:secondOperand>
    <xqx:integerConstantExpr>
      <xqx:value>1991</xqx:value>
    </xqx:integerConstantExpr>
  </xqx:secondOperand>
</xqx:greaterThanOp>
</xqx:secondOperand>
</xqx:andOp>
</xqx:whereClause>
<xqx:returnClause>
  <xqx:elementConstructor>
    <xqx:tagName>book</xqx:tagName>
    <xqx:attributeList>
      <xqx:attributeConstructor>
        <xqx:attributeName>year</xqx:attributeName>
        <xqx:attributeValueExpr>
          <xqx:pathExpr>
            <xqx:stepExpr>
              <xqx:filterExpr>
                <xqx:varRef>
                  <xqx:name>b</xqx:name>
                </xqx:varRef>
              </xqx:filterExpr>
            </xqx:stepExpr>
          <xqx:stepExpr>
            <xqx:xpathAxis>attribute</xqx:xpathAxis>
            <xqx:nameTest>year</xqx:nameTest>
          </xqx:stepExpr>
        </xqx:pathExpr>
      </xqx:attributeValueExpr>
    </xqx:attributeConstructor>
  </xqx:attributeList>
  <xqx:elementContent>
    <xqx:pathExpr>
      <xqx:stepExpr>
        <xqx:filterExpr>
          <xqx:varRef>

```

```

        <xqx:name>b</xqx:name>
      </xqx:varRef>
    </xqx:filterExpr>
  </xqx:stepExpr>
  <xqx:stepExpr>
    <xqx:xpathAxis>child</xqx:xpathAxis>
    <xqx:nameTest>title</xqx:nameTest>
  </xqx:stepExpr>
</xqx:pathExpr>
</xqx:elementContent>
</xqx:elementConstructor>
</xqx:returnClause>
</xqx:flworExpr>
</xqx:elementContent>
</xqx:elementConstructor>
</xqx:queryBody>
</xqx:mainModule>
</xqx:module>

```

## 4.6 Создание серверных сценариев на языке XQuery

Серверные сценарии в eXist позволяют создавать веб-приложения на языке XQuery. Эта технология является аналогом таких технологий, как ASP.NET, JSP, PHP.

Серверные сценарии в eXist делятся на две группы:

1. Сценарии (содержат запрос и функции). Могут быть запущены на выполнение.
2. Библиотечные модули (содержат только функции, могут быть использованы в сценариях).

### 4.6.1 Особенность использования кодировок файлов в eXist

Для того чтобы примеры можно было корректно выполнять в eXist через строку URL, необходимо чтобы файлы примеров были сохранены в кодировке UTF-8 без использования BOM.

UTF-8 является разновидностью кодировки Unicode. BOM (byte order mark) – несколько служебных байтов, которые добавляются в начало файла для определения порядка следования байтов.

Проблема состоит в том, что некоторые текстовые редакторы (в том числе стандартный блокнот Windows) автоматически добавляют BOM при сохранении текстовых файлов в формате UTF-8.

Для просмотра и редактирования примеров рекомендуется использовать редактор Notepad++, который позволяет сохранять файлы в кодировке UTF-8 без BOM.

#### 4.6.2 Простой пример серверного сценария и модуля

Для запуска следующих примеров (примеры 4.2 и 4.3) файлы данных примеров необходимо загрузить в каталог «Examples» и запускать их через строки URL в браузере:

пример 4.2 – [http://localhost:8080/exist/rest/db/Examples/module\\_1.xql](http://localhost:8080/exist/rest/db/Examples/module_1.xql)

пример 4.3 – [http://localhost:8080/exist/rest/db/Examples/module\\_2.xql](http://localhost:8080/exist/rest/db/Examples/module_2.xql)

При запуске сценариев может появляться окно авторизации, в которое необходимо ввести логин и пароль администратора.

#### Пример 4.2.

##### **module\_1.xql – пример простого сценария.**

```
xquery version "1.0";
(: Объявление версии :)

(: Возможно указание кодировки, например :)
(: xquery version "1.0" encoding "utf-8"; :)

(: ++++++ : )
(: Пролог. Объявление импортируемых модулей, опций, переменных,
функций :)
(: ++++++ : )

(: Импорт модулей, соответствующих объектам request и session :)
import module namespace request="http://exist-
db.org/xquery/request";
import module namespace session="http://exist-
db.org/xquery/session";
```

```
(: Опция, задающая вывод в виде HTML-страницы :)
declare option exist:serialize "method=xhtml media-type=text/html";

(: Объявление переменных :)
declare variable $var1 as xs:integer := 300;

(: Объявление функций :)
(: Локальные функции объявляются с префиксом local: :)
declare function local:function1($ParamUpper as xs:integer, $Base as
xs:integer) as xs:integer*
{
  for $i in 1 to $ParamUpper
    (: В функции можно использовать глобальные переменные :)
    return $var1 + $Base + $i
};

(: ++++++ : )
(: Основной запрос :)
(: ++++++ : )

<html>
  <body>
    {
      (
        for $i in 1 to 5
        return <P>{$i} - {local:function1($i,30)}</P>
      ),
      (
        for $i in 1 to 5
        return <P>{$i} - {local:function1($i,30)}</P>
      )
    }
  </body>
</html>
```

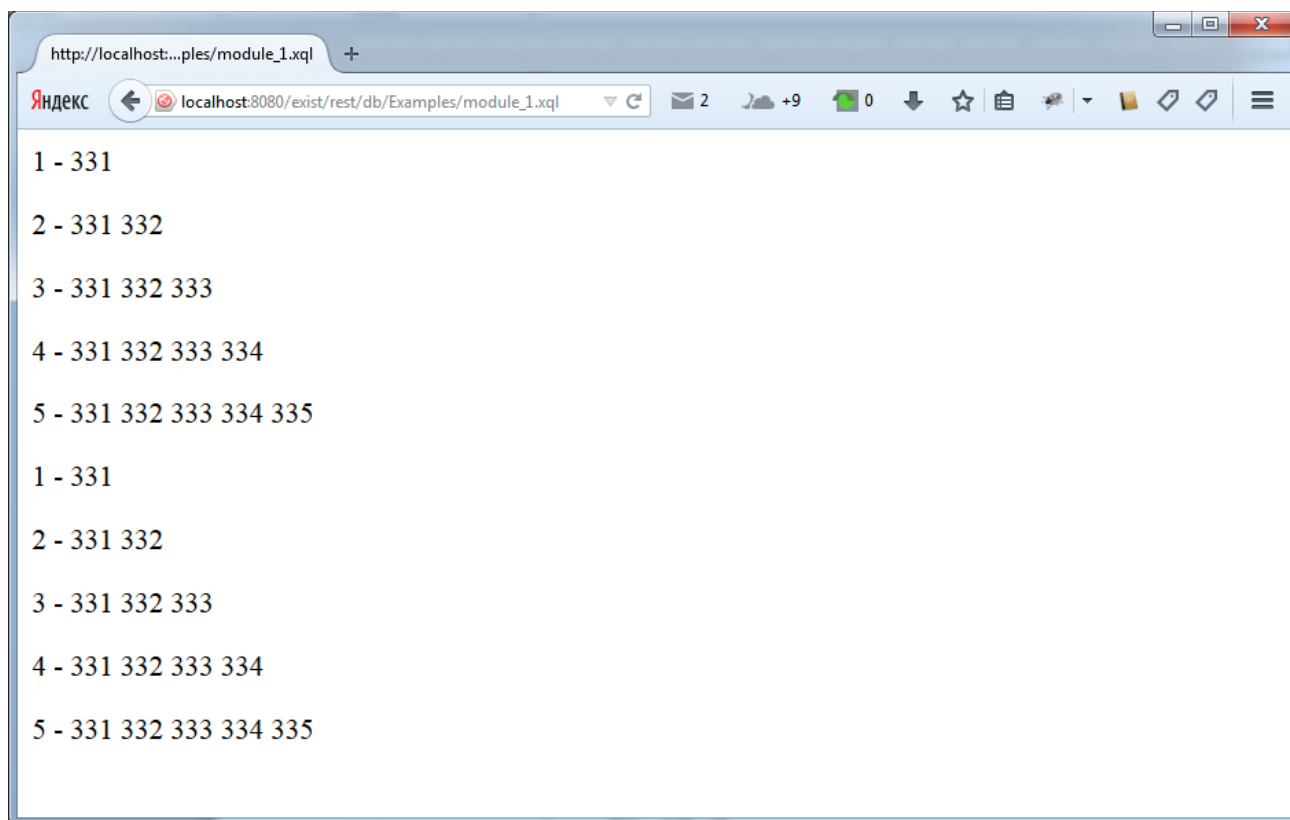
**Результат выполнения:**

Рис. 4.11. Результат выполнения примера 4.2.

**Пример 4.3.****module\_2.xql – пример сценария, использующего библиотечный модуль**

```
xquery version "1.0";
```

```
(: Объявление версии :)
```

```
(: Возможно указание кодировки, например :)
```

```
(: xquery version "1.0" encoding "utf-8"; :)
```

```
(: ++++++ :)
```

```
(: Пролог. Объявление импортируемых модулей, опций, переменных,  
функций :)
```

```
(: ++++++ :)
```

```
(: Импорт модулей, соответствующих объектам request и session :)
```

```
import module namespace request="http://exist-  
db.org/xquery/request";
```

```
import module namespace session="http://exist-
db.org/xquery/session";
```

```
(: Импорт библиотечного модуля с помощью конструкции
import module namespace префикс_пространства_имен =
"идентификатор_пространства_имен" at "путь и имя файла.xqm"; :)
```

```
import module namespace lib_module_333 = "http://lib_module_1" at
"lib_module.xqm";
```

```
(: Опция, задающая вывод в виде HTML-страницы :)
declare option exist:serialize "method=xhtml media-type=text/html";
```

```
(: ++++++ :
(: Основной запрос :)
(: ++++++ :)
```

```
<html>
```

```
<body>
```

```
{
```

```
(
```

```
for $i in 1 to 5
```

```
return <P>{$i} - {lib_module_333:function1($i,30)}</P>
```

```
(: Функция вызывается не через префикс local:, а через
префикс lib_module_333: :)
```

```
),
```

```
(<HR/>),
```

```
(: Для обращения к переменной из модуля используется префикс
:)
```

```
(<P>$lib_module_1:var1 = {$lib_module_333:var1}</P>)
```

```
(: Выражения (...), (...), (...) выполняются последовательно :)
```

```
(: Если в выражении выводится статический тэг, то он должен
быть один, например (<HR/>) :)
```

```
(: Следующая конструкция вызовет ошибку, так как в ней
выводится два тэга <HR> и <P> :)

(: (<HR/><P>$lib_module_1:var1 = {$lib_module_333:var1}</P>)
:.)

(: Возможно группировать такие конструкции в тэг верхнего
уровня :)

(: Следующая конструкция будет правильной :)

(: (<DIV> <HR/> <P>$lib_module_1:var1 =
{$lib_module_333:var1}</P> </DIV>) :)

(: Конструкция, которая выводит несколько тэгов <P> внутри
FLWOR также будет правильной,
так как FLWOR является одним выражением :)

}
</body>
</html>
```

### **lib\_module.xqm – файл библиотечного модуля.**

```
xquery version "1.0";

(: Объявление версии :)

(: Возможно указание кодировки, например :)
(: xquery version "1.0" encoding "utf-8"; :)

(: Объявление библиотечного модуля :)
module namespace lib_module_1 = "http://lib_module_1";

(: ++++++ :)
(: Пролог. Объявление импортируемых модулей, опций, переменных,
функций :)
(: ++++++ :)

(: Объявление переменных :)
(: В модуле переменные объявляются с префиксом пространства имен :)
```



```

declare variable $lib_module_1:var1 as xs:integer := 300;

(: Объявление функций :)
(: Функции объявляются не с префиксом local:, а с префиксом
объявленного пространства имен lib_module_1: :)
declare function lib_module_1:function1($ParamUpper as xs:integer,
$Base as xs:integer) as xs:integer*
{
  for $i in 1 to $ParamUpper
    (: В функции можно использовать глобальные переменные :)
    return $lib_module_1:var1 + $Base + $i
};

(: ++++++ : )
(: В библиотечном модуле нет основного запроса :)
(: ++++++ : )

```

### Результат выполнения:

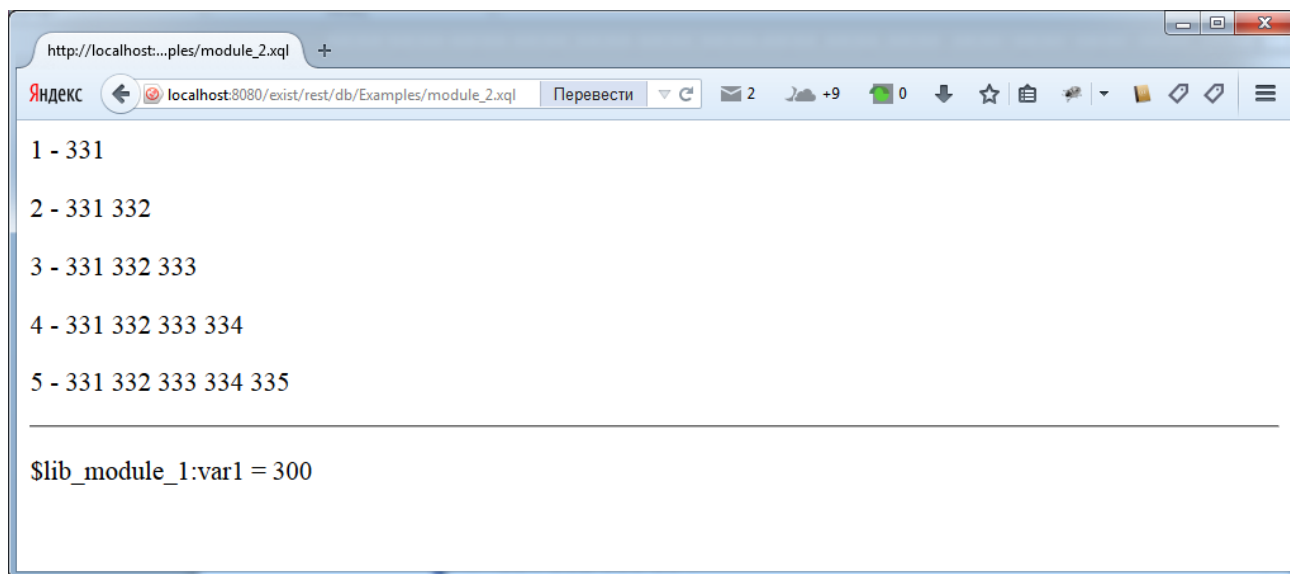


Рис. 4.12. Результат выполнения примера 4.3.

Тело функции записывается в фигурных скобках. Оно может включать в себя выражение или несколько выражений через запятую. Выражение может быть любым: FLWOR, IF, SOME, EVERY, и т.д.

Выражения, перечисляемые через запятую, выполняются последовательно.

В типе параметров функции и возвращаемого значения можно задавать количество элементов как в технологии DTD:

xs:integer – одно целое число

xs:integer? – целое число или пустое значение

xs:integer+ - последовательность целых чисел

xs:integer\* - последовательность целых чисел или пустое значение

В XQuery реализованы не все конструкции, необходимые в языке программирования. Например, нет цикла while.

### 4.6.3 Обработка документа из нескольких секций

В разделе, посвященном технологии XSLT, рассматривался пример обработки документа, состоящего из нескольких секций. Рассмотрим реализацию этого примера с использованием XQuery.

Для запуска примера необходимо загрузить файлы примера в каталог «Examples» и запускать пример через строку URL в браузере:  
<http://localhost:8080/exist/rest/db/Examples/table.xql>

#### Пример 4.4.

##### Файл table.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Языки разметки -->
<homework>
  <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>
<descriptions>

```

```

  <description language="HTML">

```

HTML (от англ. HyperText Markup Language – «язык гипертекстовой разметки») – стандартный язык разметки документов во Всемирной паутине. Большинство веб-страниц содержат описание разметки на языке HTML (или XHTML). Язык HTML интерпретируется браузерами и отображается в виде документа в удобной для человека форме.

```

  </description>

```

```

  <description language="XML">

```

XML (англ. eXtensible Markup Language – расширяемый язык разметки; произносится [экс-эм-эл]) – рекомендованный Консорциумом Всемирной паутины (W3C) язык разметки. Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому).

```

  </description>

```

```

  <description language="SGML">

```

SGML (англ. Standard Generalized Markup Language – стандартный обобщённый язык разметки; произносится [эс-джи-эм-эл]) – метаязык, на котором можно определять язык разметки для документов. SGML – наследник разработанного в 1969 году в IBM языка GML (Generalized Markup Language).

```

  </description>

```

```

</descriptions>

```

```

</homework>

```

## Файл table.xql

```
xquery version "3.0";
declare option exist:serialize "method=xhtml encoding=UTF-8 media-
type=text/html indent=no";
declare function local:main() as node()
```

Сценарий содержит функцию `main`, которая выполняет основные действия.

Название функции может быть произвольным.

```
{
<div>
  <table border="1">
    <tr>
      <th>№</th>
      <th>Язык разметки</th>
      <th>Год создания</th>
      <th>Описание</th>
    </tr>
    {
```

Обработка документа из нескольких секций осуществляется с помощью одного FLWOR-запроса.

```
let $data := doc('/db/Examples/table.xml')
```

Чтение документа из базы данных.

```
for $language in $data//language
```

В цикле перебираются все элементы `language` на произвольной глубине вложенности в документе.

```
let
  $id := xs:string($language/@id),
  $name := xs:string($language/name),
  $year := xs:string($language/year),
```

Чтение данных из текущего элемента `language` во вспомогательные переменные. Данные приводятся к строковому типу, чтобы получить текстовый узел без тэгов (например, для запроса `$language/name` результатом будет `<name>данные</name>`, а для запроса `xs:string($language/name)` результатом будет `‘данные’`).

```
$descr := xs:string($data//description[@language=$name])
```

Чтение данных из другой секции документа, поиск элемента description. Запрос `description[@language=$name]` означает, что элемент description должен содержать атрибут language совпадающий со значением переменной name.

```
order by $id ascending
```

Сортировка по возрастанию по значению переменной id.

```
return
```

```
<tr>
```

```
  <td>{$id}</td>
```

```
  <td>{$name}</td>
```

```
  <td>{$year}</td>
```

```
  <td>{$descr}</td>
```

```
</tr>
```

Элемент return формирует результирующую строку таблицы.

```
}
```

```
</table>
```

```
</div>
```

```
};
```

```
(: ++++++ :)
```

```
<html>
```

Основная часть сценария формирует структуру выходного HTML-документа.

```
<head>
```

```
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
```

```
</head>
```

```
<body>
```

```
  <h1>Обработка документа из нескольких секций</h1>
```

```
  { local:main() }
```

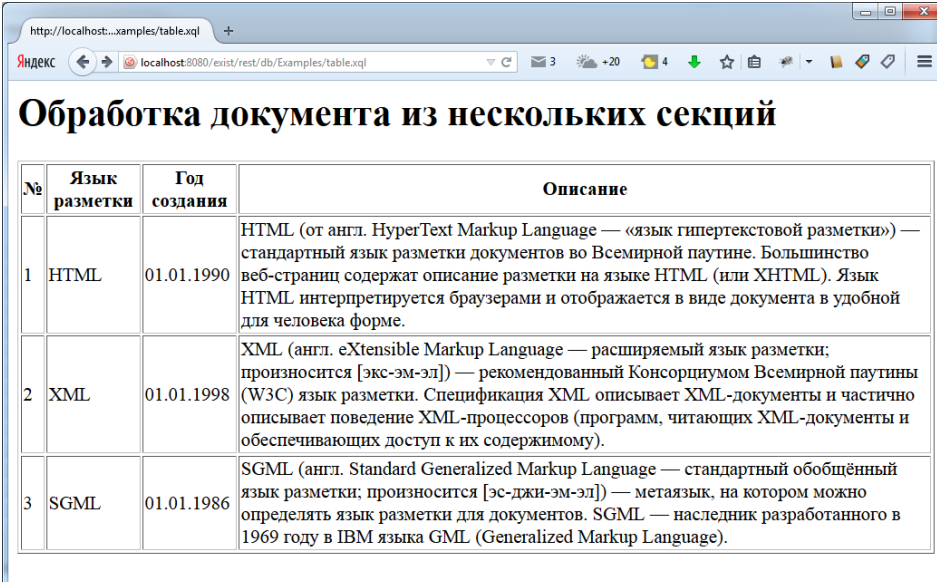
Вызов функции main.

```
</body>
```

```
</html>
```

```
(: ++++++ :)
```

**Результат выполнения:**



№	Язык разметки	Год создания	Описание
1	HTML	01.01.1990	HTML (от англ. HyperText Markup Language — «язык гипертекстовой разметки») — стандартный язык разметки документов во Всемирной паутине. Большинство веб-страниц содержат описание разметки на языке HTML (или XHTML). Язык HTML интерпретируется браузерами и отображается в виде документа в удобной для человека форме.
2	XML	01.01.1998	XML (англ. eXtensible Markup Language — расширяемый язык разметки; произносится [экс-эм-эл]) — рекомендованный Консорциумом Всемирной паутины (W3C) язык разметки. Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержанию).
3	SGML	01.01.1986	SGML (англ. Standard Generalized Markup Language — стандартный обобщённый язык разметки; произносится [эс-джи-эм-эл]) — метаязык, на котором можно определять язык разметки для документов. SGML — наследник разработанного в 1969 году в IBM языка GML (Generalized Markup Language).

Рис. 4.13. Результат выполнения примера 4.4.

Результат выполнения совпадает с аналогичным примером XSLT-преобразования.

#### 4.6.4 Пример, реализующий простую CRUD-функциональность

Рассмотрим пример, реализующий простую CRUD-функциональность. Аббревиатура CRUD (Create Read Update Delete) означает базовые операции, которые выполняются над данными: создание, чтение, обновление, удаление.

В данном примере с помощью XQuery-сценариев создаются формы просмотра, создания, обновления и удаления данных. Данные хранятся в виде XML-документа в базе данных eXist.

Пример позволяет вводить данные о наименовании СУБД и гиперссылки на сайт СУБД. Фактически данный пример является аналогом редактора реляционной таблицы, состоящей из двух полей.

Для корректной работы файлы примера необходимо разместить в коллекции «db/xrx\_simple». Пример вызывается с использованием URL: [http://localhost:8080/exist/rest/db/xrx\\_simple/index.xql](http://localhost:8080/exist/rest/db/xrx_simple/index.xql)

Пример содержит следующие сценарии и вспомогательные файлы:

- index.xql — основной сценарий, который выводит список данных;
- data.xql — сценарий, реализующий обновление и удаление данных;

- edit.xql – сценарий добавления и редактирования данных;
- module.xqm – модуль со вспомогательными функциями;
- style.css – таблица стилей, которая применяется к сгенерированным HTML-документам;

и следующие файлы данных:

- data.xml – файл с вводимыми данными;
- messages.xml – файл с сообщениями пользователю.

По умолчанию происходит обращение к сценарию index.xql, который формирует список данных.

Если пользователь нажимает кнопки «Добавить» или «Редактировать», то управление передается сценарию edit.xql, который осуществляет выдачу формы ввода или редактирования данных. Режим ввода или редактирования определяется флагом, который передается на вход сценария edit.xql.

После ввода или редактирования данных управление передается сценарию data.xql, который осуществляет добавление, обновление или удаление данных в зависимости от входных параметров, после этого происходит возврат в сценарий списка данных index.xql. Сценарий data.xql выполняет действия с данными и не выводит HTML-документов.

В случае нажатия кнопки «Удалить» происходит непосредственный вызов сценария data.xql и возврат в сценарий index.xql.

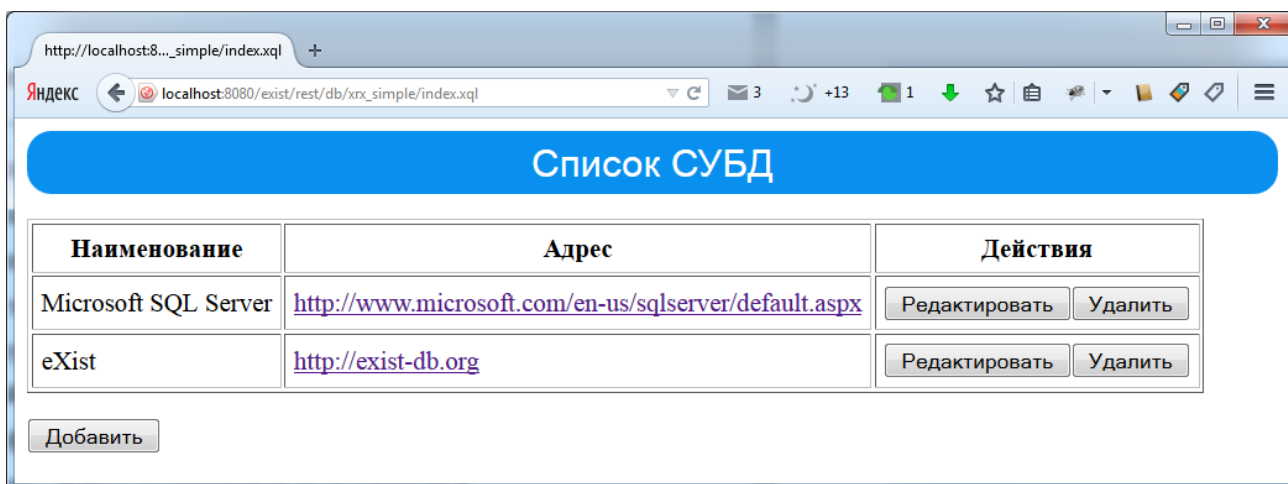


Рис. 4.14. Пример 4.5 – список данных.

Рис. 4.15. Пример 4.5 – форма добавления данных.

Рис. 4.16. Пример 4.5 – форма редактирования данных.

Рассмотрим более подробно файлы примера.

## Пример 4.5.

### Файл data.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<db>
  <db>
    <id>1</id>
    <name>Microsoft SQL Server</name>
    <uri>http://www.microsoft.com/en-
us/sqlserver/default.aspx</uri>
  </db>
  <db>
    <id>2</id>
    <name>eXist</name>
    <uri>http://exist-db.org</uri>
  </db>
```

[Оглавление](#)



```
</db>
```

### Файл messages.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<messages>
  <mes key="index_list">Список СУБД</mes>
  <mes key="index_list_actions">Действия</mes>
  <mes key="index_list_data_name">Наименование</mes>
  <mes key="index_list_data_uri">Адрес</mes>
  <mes key="edit_header_new">Добавление</mes>
  <mes key="edit_header_edit">Редактирование</mes>
  <mes key="form_add_new">Добавить</mes>
  <mes key="form_edit">Редактировать</mes>
  <mes key="form_del">Удалить</mes>
  <mes key="form_save">Сохранить</mes>
  <mes key="form_exit">Отменить</mes>
</messages>
```

### Файл module.xqm

```
xquery version "3.0";
module namespace xrx_example =
"http://iu5.bmstu.ru/edu/xrx_example";
```

Объявление пространства имен модуля.

```
declare namespace xdb="http://exist-db.org/xquery/xmlldb";
```

```
declare variable $xrx_example:IndexPath {"index.xml"};
declare variable $xrx_example>EditPath {"edit.xml"};
declare variable $xrx_example:DataPath {"data.xml"};
declare variable $xrx_example:DataStorePath {"data.xml"};
```

Объявление переменных с названиями сценариев, переменные используются для чтения файлов данных из eXist и формирования гиперссылок.

```
declare function xrx_example:getMessage($key as xs:string) as
xs:string?
```

Функция получения строки с сообщением для пользовательского интерфейса. Параметром является ключ сообщения.

```
{
    let $mes_file := doc("messages.xml")
    let $mes := $mes_file//mes[@key=$key]
    return $mes
};
```

declare function xrx\_example:login()

Функция входа в БД с правами администратора. Используется для выполнения запросов.

```
{
    xdb:register-database("org.exist.xmldb.DatabaseImpl", true()),
    let
        $User := "admin",
        $Password := ""
    return xdb:login("xmldb:exist:///db", $User, $Password)
};
```

### Файл index.xql

```
xquery version "3.0";
declare option exist:serialize "method=xhtml encoding=UTF-8 media-
type=text/html indent=no";
```

Выходным форматом сценария является HTML-документ в кодировке UTF-8.

```
import module namespace xrx_example =
"http://iu5.bmstu.ru/edu/xrx_example" at "module.xqm";
```

declare function local:main() as node()

Функция формирования таблицы данных.

```
{
<div>
<table border="1">
<tr>
    <th>{xrx_example:getMessage("index_list_data_name")}</th>
```

[Оглавление](#)

```
<th>{xrx_example:getMessage("index_list_data_uri")}</th>
<th>{xrx_example:getMessage("index_list_actions")}</th>
```

Наименования колонок таблицы выбираются из файла с сообщениями с помощью функции `xrx_example:getMessage`, функция объявлена в модуле `module.xqm`. В качестве параметра функция принимает ключ сообщения.

```
</tr>
```

```
{
```

```
let $data := doc($xrx_example:DataStorePath)
```

В переменную `data` считывается файл данных из `eXist`.

```
for $d in $data//db
```

В цикле перебираются данные из элементов `db`.

```
let
```

```
$id := xs:string($d/id),
```

Id элемента данных.

```
$name := xs:string($d/name),
```

Поле данных «Наименование СУБД».

```
$uri := xs:string($d/uri)
```

Поле данных «Гиперссылка на сайт СУБД».

```
order by $name ascending
```

Сортировка по полю «Наименование СУБД» по возрастанию.

```
return
```

Формирование строки таблицы с результирующими данными.

```
<tr>
```

```
<td>{$name}</td>
```

Вывод поля данных «Наименование СУБД».

```
<td><a target="_blank" href="{ $uri }">{$uri}</a></td>
```

Вывод в виде гиперссылки поля данных «Гиперссылка на сайт СУБД».

```
<td>
```

```
<form class="form_compact" method="post"
```

```
action="{ $xrx_example:EditPath }">
```

Форма для вызова сценария редактирования данных. Поле `action` содержит ссылку на сценарий, отвечающий за создание и редактирование данных.

```
<input type="hidden" name="id" value="{ $id}" />
```

Поле, содержащее id элемента для редактирования.

```
<button  
type="submit">{xrx_example:getMessage("form_edit")}</button>
```

Кнопка «Редактировать» для отправки формы на сервер.

```
</form>
```

```
<form class="form_compact" method="post"  
action="{ $xrx_example:DataPath}">
```

Форма для вызова сценария удаления данных. Поле action содержит ссылку на сценарий, отвечающий за обновление и удаление данных.

```
<input type="hidden" name="id" value="{ $id}" />
```

Поле, содержащее id элемента для редактирования.

```
<input type="hidden" name="del" value="1" />
```

Флаг удаления данных.

```
<button  
type="submit">{xrx_example:getMessage("form_del")}</button>
```

Кнопка «Удалить» для отправки формы на сервер.

```
</form>
```

```
</td>
```

```
</tr>
```

```
}
```

```
</table>
```

```
<p>
```

```
<form method="post" action="{ $xrx_example>EditPath}">
```

Форма для вызова сценария создания данных.

```
<input type="hidden" name="new" value="1" />
```

Флаг создания данных.

```
<button  
type="submit">{xrx_example:getMessage("form_add_new")}</button>
```

Кнопка «Добавить» для отправки формы на сервер.

```
</form>
</p>
```

```
</div>
};
```

```
(: ++++++ :)
```

```
<html>
```

**Основная часть сценария формирует структуру выходного HTML-документа.**

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8"/>
  <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
  <h1>{xrx_example:getMessage("index_list")}</h1>
  { local:main() }
</body>
</html>
(: ++++++ :)
```

## **Файл edit.xql**

```
xquery version "3.0";
declare option exist:serialize "method=xhtml encoding=UTF-8 media-
type=text/html indent=no";
import module namespace xrx_example =
"http://iu5.bmstu.ru/edu/xrx_example" at "module.xqm";

declare function local:main($newParam as xs:string, $IdParam as
xs:string) as node()
```

**Функция формирования формы ввода или редактирования данных.**

```
{
let
  $data := doc($xrx_example:DataStorePath),
  $d := $data//db[id=$IdParam],
```

[Оглавление](#)

```
$default_name :=
    if($IdParam="0")
    then ("" )
    else ($d/name),
```

```
$default_uri :=
    if($IdParam="0")
    then ("" )
    else ($d/uri)
```

Если `IdParam="0"`, то это признак создания нового элемента, иначе из `eXist` читаются текущие значения полей данных.

```
return
```

```
<div>
```

```
<form method="post" action="{ $xrx_example:DataPath }">
```

Содание формы, которая ссылается на сценарий `data.xql`.

```
<input type="hidden" name="id" value="{ $IdParam }" />
```

Скрытое поле `id` элемента.

```
<input type="hidden" name="new" value="{ $newParam }" />
```

Скрытое поле добавления данных.

```
<input type="hidden" name="save" value="1" />
```

Скрытое поле сохранения данных.

```
<table align="center" border="1">
```

```
<tr>
```

```
<td>{xrx_example:getMessage("index_list_data_name")}:</td>
```

```
<td>
```

```
<input type="text" size="50" name="data_name"
value="{ $default_name }" />
```

Редактор поля данных «Наименование СУБД». Если установлен режим добавления данных, то поле редактирования пустое, иначе оно содержит текущее значение из файла данных eXist.

```

        </td>
    </tr>

    <tr>
        <td>{xrx_example:getMessage("index_list_data_uri")}</td>
        <td>
            <input type="text" size="50" name="data_uri"
value="{ $default_uri}" />

```

Редактор поля данных «Гиперссылка на сайт СУБД». Если установлен режим добавления данных, то поле редактирования пустое, иначе оно содержит текущее значение из файла данных eXist.

```

        </td>
    </tr>

    <tr>
        <td colspan="2" align="center">
            <button
type="submit">{xrx_example:getMessage("form_save")}</button>

```

**Кнопка сохранения отправляет форму сценарию data.xql.**

```

            <button type="button"
onclick="window.location='{ $xrx_example:IndexPath }'">{xrx_example:ge
tMessage("form_exit")}</button>

```

**Кнопка отмены реализует возврат в форму списка данных.**

```

        </td>
    </tr>
</table>
</form>
</div>
};

```

```
(: ++++++ :)
```

```
let
```

```
$newParam := xs:string(request:get-parameter("new","0")),
```

```
$IdParam := xs:string(request:get-parameter("id","0"))
```

**Id** элемента читается из параметров сценария.

```
return
```

```
<html>
```

**Основная часть сценария формирует структуру выходного HTML-документа.**

```
<head>
```

```
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-
```

```
8"/>
```

```
  <link rel="stylesheet" type="text/css" href="style.css"/>
```

```
</head>
```

```
<body>
```

```
  {
```

```
    let $HeaderMessage :=
```

```
    if($newParam="1")
```

```
    then (xrx_example:getMessage("edit_header_new"))
```

```
    else (xrx_example:getMessage("edit_header_edit"))
```

```
    return <h1>{$HeaderMessage}</h1>
```

**В зависимости от режима выводится заголовок формы «Добавление» или «Редактирование».**

```
  }
```

```
  { local:main($newParam, $IdParam) }
```

```
</body>
```

```
</html>
```

## **Файл data.xql**

```
xquery version "3.0";
```

```
import module namespace xrx_example =
```

```
"http://iu5.bmstu.ru/edu/xrx_example" at "module.xqm";
```



```
declare function local:NewData($newParam as xs:string, $IdParam as
xs:string, $data_name as xs:string, $data_uri as xs:string) as
node()
```

**Функция формирования структуры XML-данных для добавления новых данных.**

```
{
let
    $id :=
        if($newParam='1')
        then (util:uuid())
        else ($IdParam)
return
    <db>
        <id>{$id}</id>
        <name>{$data_name}</name>
        <uri>{$data_uri}</uri>
    </db>
};
```

```
declare function local:SaveData($newParam as xs:string, $IdParam as
xs:string, $data_name as xs:string, $data_uri as xs:string) as
node() *
```

**Функция сохранения данных в файле data.xml.**

```
{
    let $data := local:NewData($newParam, $IdParam, $data_name,
$data_uri)
    return
        if($newParam='1')
```

**Если установлен флаг добавления данных.**

```
    then
        (
            update insert $data
            into doc($xrx_example:DataStorePath)//dbs
```

)

То добавление элемента данных.

else

(

update replace

doc(\$xrx\_example:DataStorePath)//db[id=\$IdParam]

with \$data

Иначе обновление элемента данных.

)

};

declare function local:DelData(\$IdParam as xs:string) as node()\*

Функция удаления данных.

{

let \$data := doc(\$xrx\_example:DataStorePath)//db[id=\$IdParam]

return update delete \$data

};

Основной сценарий:

let

\$newParam := xs:string(request:get-parameter("new","0")),

\$saveParam := xs:string(request:get-parameter("save","0")),

\$delParam := xs:string(request:get-parameter("del","0")),

\$IdParam := xs:string(request:get-parameter("id","0")),

\$data\_name := xs:string(request:get-parameter("data\_name","")),

\$data\_uri := xs:string(request:get-parameter("data\_uri",""))

Установка флагов, которые передаются как параметры URL.

return

(

xrx\_example:login(),

Вход в eXist с правами администратора для изменения данных.

(

if(\$saveParam = '1')

then

```
(
    local:SaveData($newParam, $IdParam, $data_name, $data_uri)
```

Если установлен флаг сохранения, то выполняется сохранение данных.

Выполняется добавление или обновление в зависимости от флагов.

```
)
else
(
    if($delParam = '1')
    then
    (
        local:DelData($IdParam)
    )
    else ()
)
),
response:redirect-to(xs:anyURI($xrx_example:IndexPath))
    Передача управления сценарию index.xql.
)
```

Данный пример является простым XRX-приложением. Более подробно архитектура XRX рассматривается в следующих главах пособия.

## **4.7 XQuery и XSLT**

В качестве подведения итогов по данной главе хотелось бы сравнить некоторые особенности XQuery и XSLT.

На первый взгляд может показаться, что технология XQuery полностью заменяет XSLT. Действительно, многие задачи по обработке документов XML решаются проще с использованием XQuery, и код на XQuery может получиться более понятным и компактным чем код на XSLT.

Однако хотелось бы обратить внимание на следующие особенности данных технологий.

Прямым аналогом XQuery является не XSLT, а язык XPath, встроенный в XSLT. Существует неформальное соглашение о том, что XQuery и XPath должны быть использованы для обработки XML-документов, а XSLT для преобразования XML-документов в HTML-документы. На практике это неформальное соглашение часто нарушается, и XQuery-сценарии напрямую генерируют HTML-документы.

Важной особенностью XSLT является конструкция `xsl:apply-templates`, аналога которой в XQuery не существует. XQuery не позволяет адаптироваться к структуре обрабатываемого XML-документа.

Поэтому языки XQuery и XPath в большей степени являются языками запросов к XML-данным, а технология XSLT в большей степени технологией преобразования данных, которая позволяет адаптироваться к структуре обрабатываемого XML-документа. При этом существует довольно много задач по обработке XML-документов, которые можно решить как с использованием XQuery, так и с использованием XSLT+XPath 1.0.

#### **4.8 Материалы для дальнейшего изучения**

Рекомендуется ознакомиться со спецификациями XQuery 1.0 [XQuery, 2007], и XQuery 3.0 [XQuery, 2014].

Полный список функций и операторов XQuery 1.0 и 3.0 приведены в спецификациях [XQuery Functions, 2007], [XQuery Functions, 2014]. Эта спецификация включает порядка сотни функций и операторов, в примерах были рассмотрены лишь некоторые из них.

Наиболее подробной книгой по языку XQuery на русском языке является [XQuery руководство, 2005].

#### **4.9 Контрольные вопросы**

1. Что такое естественные и приспособленные XML-ориентированные СУБД?
2. Как хранятся XML-данные в СУБД eXist?

[Оглавление](#)

3. В чем основные особенности модели данных, используемой в XQuery?
4. Как работать с последовательностями в XQuery?
5. Как устроены операторы сравнения в XQuery?
6. Как использовать конструкторы элементов в XQuery?
7. Как использовать оператор FLWOR в XQuery?
8. Как можно соединять документы с помощью оператора FLWOR в XQuery?
9. Как используются выражения `some` и `every` в XQuery?
10. Какие действия над множествами используются в XQuery?
11. Как используются выражения `ordered` и `unordered` в XQuery?
12. Какие выражения для работы с типами данных используются в XQuery?
13. Как производится обновление данных в БД с использованием оператора FLWOR в XQuery?
14. Какие новые возможности появились в XQuery 3.0?
15. Как создавать серверные сценарии в eXist с использованием XQuery?
16. Как создавать библиотечные модули в eXist с использованием XQuery?

## 5 Часть 5. Технология XForms

Технология XForms предназначена для ввода данных в формате XML. Она является развитием технологии HTML-форм.

### 5.1 Основные отличия XForms-форм от HTML-форм

В HTML-формах:

1. Элементы управления формы (поля ввода, списки и т.д.) задаются в виде HTML-тэгов.
2. Значения по умолчанию для данных записываются в виде атрибута value этих тэгов. То есть значения по умолчанию для данных "встроены" в поля ввода.
3. Заполненные данные передаются на веб-сервер в виде набора пар "имя=значение". Формат данных "на выходе" формы не совпадает с форматом, который используется для задания значений по умолчанию.

Эти особенности не стоит считать недостатками технологии HTML-форм. Это "классический" подход, который используется в большинстве технологий веб-разработки.

XForms-формы являются более «однородным» решением:

1. Элементы управления формы (как и в случае HTML-форм) задаются в виде тэгов (XForms).
2. Данные хранятся в специальной "модели", которой не было в HTML. Для хранения данных используется XML-фрагмент, вложенные XML-элементы соответствуют полям ввода. Кроме того, в модели можно хранить типы данных и ограничения, которые должны учитываться при вводе данных. Эти ограничения проверяются XForms-процессором при вводе данных.
3. Значения по умолчанию задаются в виде XML-фрагмента в модели данных. В процессе редактирования формы этот XML-фрагмент

изменяется. То есть формат данных "на выходе" формы совпадает с форматом, который используется для задания значений по умолчанию.

В технологии XForms и сами данные, и ограничения на данные, и поля ввода данных задаются в виде XML.

## **5.2 Особенности технологии XForms**

1. XForms-форма не является «самостоятельным» документом. Она должна быть строена в основной документ (host document), который разработан на основном языке разметки (host language). Как правило, в качестве такого языка используется XHTML.

Обычно модель данных располагается в секции HEAD документа XHTML, а элементы управления формы в секции BODY.

2. В XForms используется «конкретная» модель и «абстрактные» элементы управления формы.

Модель данных «конкретная», потому что она разрабатывается очень детально, задаются типы данных, ограничения на данные. Все ограничения на данные задаются не в полях ввода, а в модели.

Элементы управления формы являются «абстрактными», потому что нет точных правил для их отображения. XForms-процессор определяет, как отображать элементы управления.

3. Проверка правильности введенных данных осуществляется при вводе данных формы на стороне «клиента», а не стороне «сервера» (как в «традиционных» веб-приложениях).

4. Данные по умолчанию для формы и введенные данные представляют собой XML-фрагмент, а не набор отдельных «переменных» (как в HTML-формах).

Это позволяет проверить вводимые данные на стороне клиента и после ввода данных «в одно действие» сохранить полученный XML-фрагмент с проверенными данными в XML-ориентированную БД.

5. В технологии XForms в значительной степени используется язык XPath (для связи элементов, задания ограничений и т.д.). В настоящее время используется XPath 1.0.

### **5.3 Программные продукты для работы с XForms**

Основным программным продуктом для работы с XForms является XForms-процессор, который «выполняет» форму: читает форму, загружает данные по умолчанию, отображает элементы управления, позволяет вводить данные и проверяет правильность ввода данных, передает введенные данные на указанный URI.

Несмотря на «абстрактность» элементов управления, большинство XForms-процессоров являются веб-приложениями и превращают XForms-формы в обычные HTML-формы (как правило, с использованием CSS, JavaScript, AJAX).

Из свободно-распространяемых XForms-процессоров можно выделить четыре:

1. Orbeon Forms.
2. betterFORM.
3. XSLTForms.
4. Mozilla XForms Project.

Рассмотрим эти XForms-процессоры более подробно.

#### **5.3.1 XForms-процессор «Orbeon Forms»**

Orbeon Forms (<http://www.orbeon.com>) является одним из самых развитых XForms-процессоров.

Реализован на Java в виде веб-приложения.

Поставляется вместе со встроенным экземпляром «eXist». Содержит визуальный редактор XForms-форм. Также содержит язык XPL (XML Pipeline Definition Language), который позволяет задавать workflow для обработки XML-данных. Подробно документирован.



Orbeon Forms является достаточно сложным продуктом с большим объемом документации. Для разработки серьезных XForms-приложений необходимо использовать коммерческую версию.

### 5.3.2 XForms-процессор «betterFORM»

betterFORM (<http://www.betterform.de>) является более простым решением по сравнению с Orbeon Forms. Содержит только XForms-процессор. Является развитием XForms-процессора «Chiba».

Реализован на Java в виде веб-приложения.

Далее в этой главе мы будем использовать именно betterFORM для работы с учебными примерами.

Для работы с Orbeon Forms или betterFORM необходимо использовать какой-либо веб-сервер на основе Java-платформы (например, Apache Tomcat).

### 5.3.3 XForms-процессор «XSLTForms»

XForms-процессор XSLTForms (<http://www.agencexml.com/xsltforms>) является «клиентским» решением и работает в браузере (нет необходимости использовать веб-сервер).

Реализован с использованием XSLT, CSS, JavaScript.

Основным элементом процессора XSLTForms является XSLT-преобразование, с помощью которого необходимо преобразовать XHTML-файл, содержащий XForms-форму, в HTML. Преобразование можно сделать и в браузере, и в серверном сценарии.

XSLTForms встроен в последние версии «eXist». XSLTForms можно достаточно просто использовать в том случае, когда XForms-форма динамически генерируется в «eXist» с помощью серверного XQuery-сценария.

При разработке XRX-приложения мы будем использовать процессор XSLTForms.

### 5.3.4 XForms-процессор «Mozilla XForms»

Этот XForms-процессор (<http://www.mozilla.org/projects/xforms>) является плагином к Firefox. То есть он, как и XSLTForms, также является «клиентским» решением. В настоящее время компания Mozilla отказалась от поддержки данного плагина.

### 5.3.5 Преобразование XML-схем в XForms-формы

Существуют продукты (библиотеки), предназначенные для автоматизированного преобразования XML-схем в XForms-формы.

1. Библиотека XSDTransformer (<http://xsdtrans.sourceforge.net>).
2. Библиотека XSLTKit (<http://xsltkit.sourceforge.net>).

Все средства вызываются из консоли и преобразуют файл XML-схемы в форму XForms.

Библиотеки XSDTransformer и XSLTKit представляют собой XSLT-преобразования для решения различных задач. В том числе это задача преобразования XML-схемы в документ XForms.

Однако, такие продукты применяются не очень часто по следующим причинам. Во-первых, они обычно генерируют «полуфабрикат» формы, который необходимо исправлять вручную, во-вторых, задача генерации формы на основе схемы может встречаться достаточно редко.

### 5.3.6 Microsoft InfoPath

Microsoft InfoPath является альтернативой технологии XForms. Позволяет создавать формы ввода, вводить данные и сохранять данные в формате XML.

Технология XForms при этом не используется. Формы хранятся в специальном формате «Microsoft InfoPath».

## **5.4 Работа с примерами в этой главе**

В этой главе все примеры XForms-форм созданы в виде документов XHTML. Это связано с тем, что XForms не является самостоятельным языком разметки, но XForms-форма должна быть встроена в основной документ (host document), который разработан на основном языке разметки (host language). Здесь в качестве host-языка используется XHTML.

Примеры в этой главе ориентированы на использование XForms-процессора betterFORM.

Для запуска примеров необходимо скачать и установить процессор betterFORM (<http://www.betterform.de>). Рекомендуется использовать дистрибутив в виде веб архива (.war) и установить его на Apache Tomcat. Веб-приложение будет установлено в каталог «betterform».

Рекомендуется использовать Apache Tomcat версии 7, в версии 8 были замечены проблемы при развертывании веб архива.

После запуска Apache Tomcat необходимо обратиться по адресу <http://localhost:8080/betterform>

В результате обращения будет выдан экран загрузки XForms-форм, представленный на следующем рисунке.

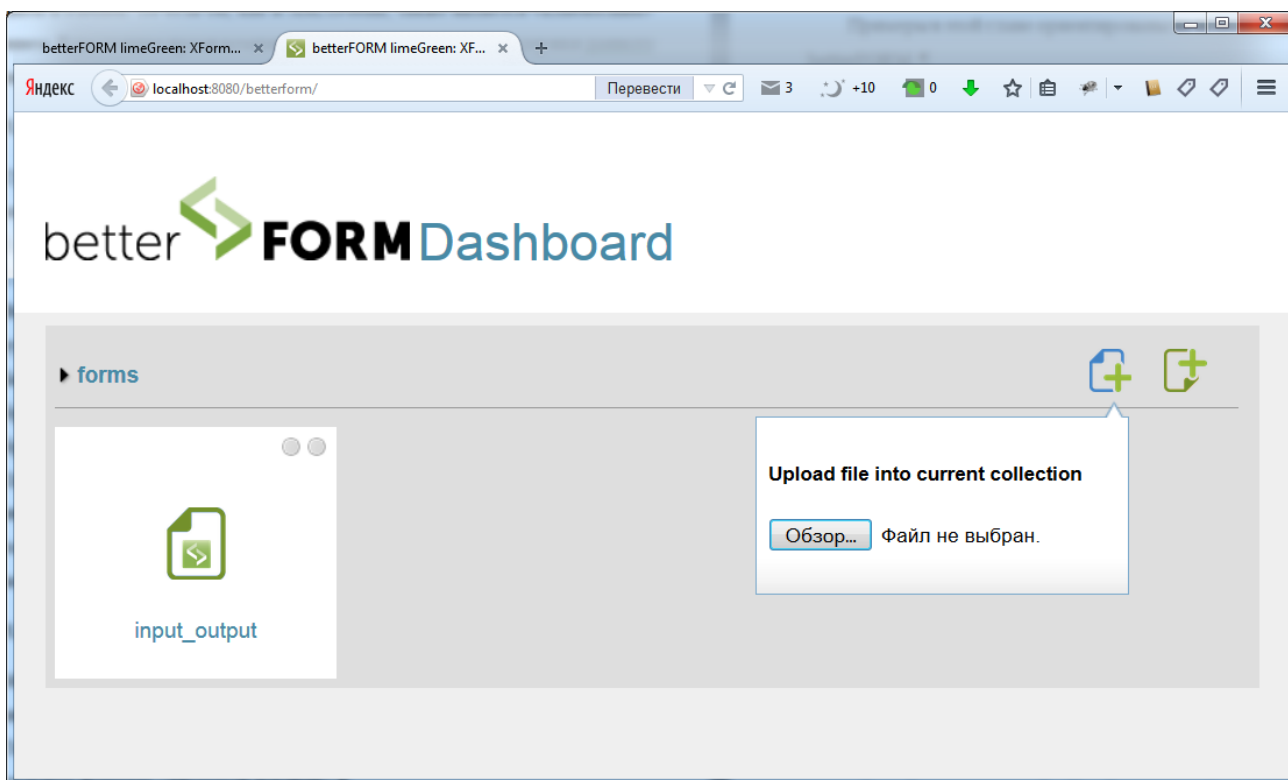


Рис. 5.1. Загрузка форм в приложение betterform.

При нажатии на кнопку обзор открывается стандартный диалог загрузки файла. Необходимо загрузить документы в формате xhtml из примеров к данной главе.

После загрузки пример отображается в галерее и может быть запущен двойным нажатием мыши.

## 5.5 Базовые примеры XForms-форм

### Пример 5.1. Файл «input\_output.xhtml».

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
```

Пространством имен по умолчанию является пространство имен для XHTML-документов «http://www.w3.org/1999/xhtml».

Также объявляется пространство имен для XForms – `xmlns:xforms="http://www.w3.org/2002/xforms"`. В качестве префикса пространства имен обычно используется «xforms» или «xf».

Пространство имен `xmlns:ev="http://www.w3.org/2001/xml-events"` используется в том случае, когда необходимо создавать динамические формы с реакцией на события (например, нажатие на кнопку).

В этом примере события не используются (стандартная кнопка отправки формы не требует использования событий).

```
<head>
  <title>Пример ввода и вывода значения</title>

  <!-- Пустой префикс пространства имен обязателен для
корневого элемента xforms:instance -->
  <xforms:model>
```

Элемент «xforms:model» объявляет модель данных формы. Модель данных формы объявляется в секции HEAD документа XHTML.

Модель может содержать:

1. Экземпляры данных (xforms:instance).
2. Ограничения на данные (xforms:bind).
3. Способы отправки заполненных данных на сервер (xforms:submission).

```
<xforms:instance xmlns="">
```

Элемент «xforms:instance» объявляет экземпляр данных.

В этом примере экземпляр данных непосредственно вложен в элемент «xforms:instance». С помощью атрибута «src=URI экземпляра данных» элемента «xforms:instance» можно загружать экземпляр данных из внешнего URI (например, из серверного сценария).

Атрибут «xmlns=""» указывает, что для данных используется «пустое» пространство имен (то есть в экземпляре данных не используются пространства имен).

```
<example>
  <text/>
```

```
</example>
```

Экземпляр данных содержит корневой элемент экземпляра данных «example» и элемент данных «text», который в дальнейшем будет связан с полем ввода.

```
</xforms:instance>
```

Атрибут «xmlns=""» можно перенести из xforms:instance в корневой элемент экземпляра данных.

В этом случае вместо

```
<xforms:instance xmlns="">
```

```
  <example>
```

```
    <text/>
```

```
  </example>
```

```
</xforms:instance>
```

будет

```
<xforms:instance >
```

```
  <example xmlns="">
```

```
    <text/>
```

```
  </example>
```

```
</xforms:instance>
```

Но, как правило, пустое пространство имен удобнее объявлять в элементе «xforms:instance».

```
<!-- submission указывает куда необходимо отправить форму -->
```

```
<xforms:submission id="submit_id" action="" method="post"
```

```
includenamespacesprefixes=""/>
```

Элемент «xforms:submission» определяет правила отправки формы на сервер.

В нашем примере элемент «xforms:submission» содержит только основные атрибуты. Обязательный атрибут «id» используется для ссылки на элемент «xforms:submission».

Атрибут «action» задает URI серверного сценария, которому отправляются данные (в нашем примере он пустой, данные не отправляются).

Атрибут «method» задает метод HTTP-протокола для отправки данных (обычно используется POST, иногда PUT). Метод GET здесь не может быть использован, так как XML-данные не должны передаваться через строку URI.

Атрибут «includenamespacesprefixes» позволяет включать в передаваемые данные нужные префиксы пространств имен. Это полезно в том случае, когда данные содержат пространства имен. Пустое значение атрибута указывает, что префиксы пространств имен не передаются на сервер.

```
</xforms:model>
</head>
```

```
<body>
```

Поля ввода данных объявляются в секции BODY документа XHTML.

```
<xforms:group>
```

В простом случае поля ввода данных размещаются внутри контейнерного элемента «xforms:group».

```
<!-- Элемент input создает поле ввода, label
определяет подпись к полю ввода -->
<xforms:input ref="text">
    <xforms:label>Введите текст:</xforms:label>
</xforms:input>
```

Элемент «xforms:input» определяет поле ввода данных, в атрибуте «ref» указывается XPath-выражение, которое ссылается на XML-элемент данных в экземпляре модели.

В нашем случае поле ввода соответствует XML-элементу «text».

В XPath-выражении (атрибут ref) не учитывается корневой элемент модели (example), то есть атрибут ref="text", а не ref="example/text".

Элемент «xforms:label» содержит подпись к полю ввода.

```
<!-- Вывод текста из элемента text. В атрибуте value
используется XPath-выражение -->
<xforms:output value="concat('!!! ', text, ' !!!')">
    <xforms:label>Вывод текста:</xforms:label>
</xforms:output>
```

Элемент «`xforms:output`» используется для вывода вычисляемых значений. В атрибуте «`value`» задается вычисляемое значение в виде XPath-выражения.

В нашем случае с помощью функции `concat` соединяется несколько строк, одной из которых является значение XML-элемента `text`.

Если мы введем значение в поле ввода, то в инстансе модели изменится XML-элемент `text` и содержимое `xforms:output` будет автоматически пересчитано.

```

<!-- Кнопка отправки формы. Атрибут submission
ссылается на элемент xforms:submission в модели -->
<xforms:submit submission="submit_id">
    <xforms:label>Отправка формы</xforms:label>
</xforms:submit>

```

Элемент «`xforms:submit`» создает кнопку отправки формы на сервер. Атрибут «`submission`» ссылается на атрибут «`id`» соответствующего элемента «`xforms:submission`» в модели.

Элемент «`xforms:label`» может по-разному отображаться в зависимости от того, в какой элемент он вложен. Если он вложен в поле ввода, то отображается как подпись к полю ввода, если он вложен в кнопку, то отображается как надпись на кнопке.

```

</xforms:group>
</body>
</html>

```

Результат выполнения примера в betterFORM:

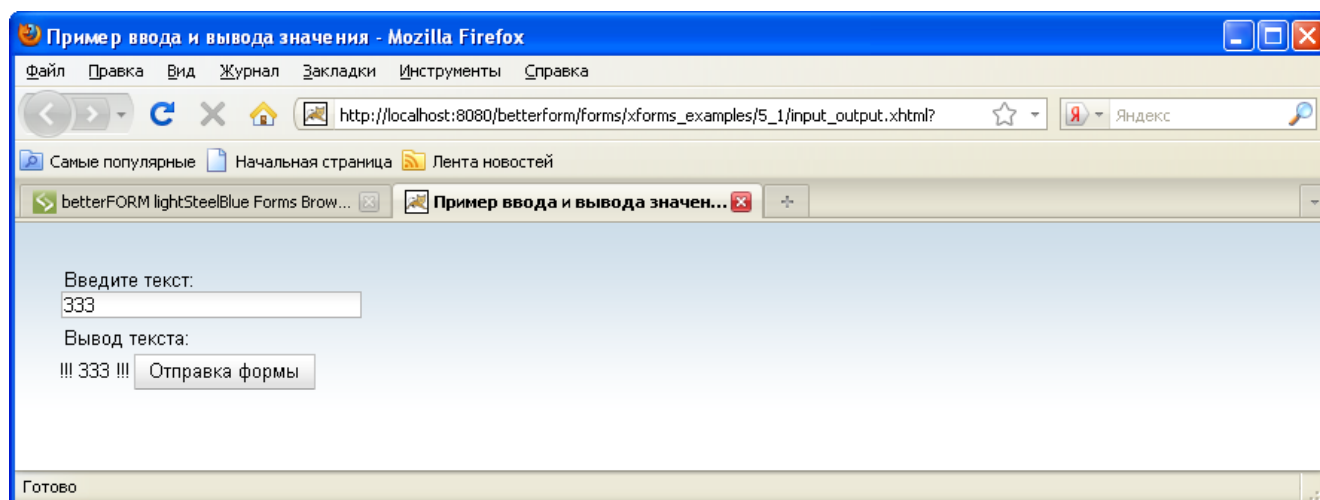




Рис. 5.2. Результат выполнения примера 5.1. Файл «input\_output.xhtml».

Как и обычная HTML-форма, XForms-форма может быть вложена в табличные тэги.

Следующий пример является модификацией предыдущего. Модель данных не изменилась, но поля ввода формы размещены в табличных тэгах.

### Пример 5.1. Файл «input\_output\_table.xhtml».

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример ввода и вывода значения с таблицей</title>

    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model>
      <xforms:instance>
        <example xmlns="">
          <text/>
        </example>
      </xforms:instance>
      <xforms:submission action="" method="post" id="submit_id"
includenamespaceprefixes=""/>
    </xforms:model>
  </head>

  <body>
    <table border="2" cellspacing="5">
    <tr>
      <td><b>Введите текст:</b></td>

      <!-- Элемент input пустой (не содержит элемента label), текст
подсказки размещается в ячейке таблицы -->
      <td><xforms:input ref="text"/></td>
```

```

</tr>
<tr>
  <td><b>Вывод текста:</b></td>
  <td><xforms:output value="concat('!!! ', text, ' !!!')"/></td>
</tr>
<tr>
  <td colspan="2" align="center">
    <xforms:submit submission="submit_id">
      <xforms:label>Отправка формы</xforms:label>
    </xforms:submit>
  </td>
</tr>
</table>
</body>
</html>

```

Результат выполнения примера:

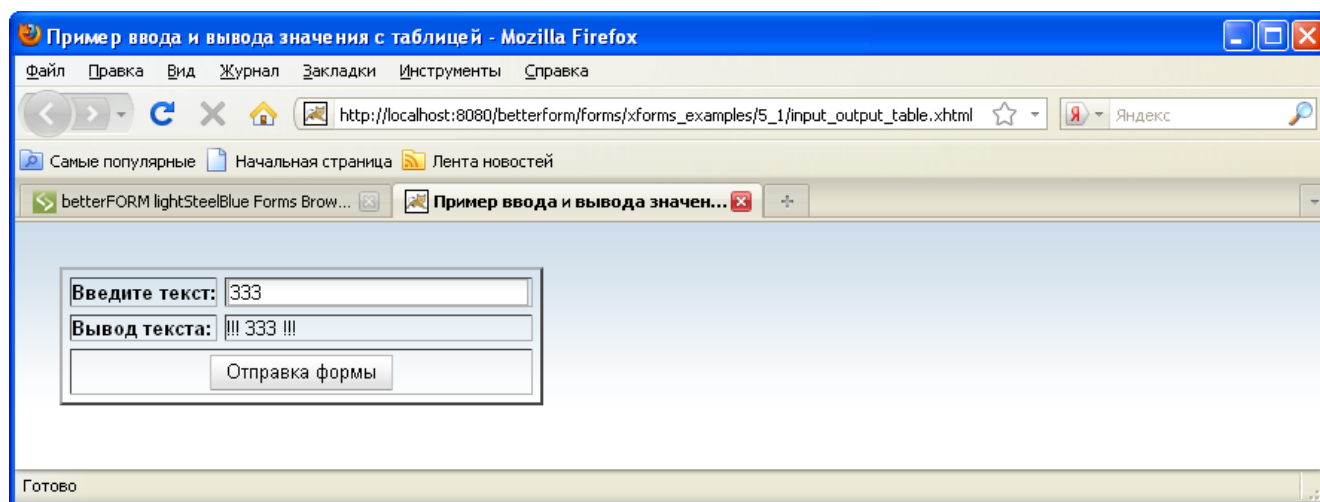


Рис. 5.3. Результат выполнения примера 5.1. Файл «input\_output\_table.xhtml».

## 5.6 Модель данных формы

Рассмотрим примеры, в которых используются более сложные модификации модели данных формы.

### 5.6.1 Задание ограничений на типы данных

В следующем примере с помощью элементов «xforms:bind» задаются типы данных для XML-данных в инстансе.

#### Пример 5.2. Файл «input.xhtml».

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример элемента input и элемента bind</title>

    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model>
      <xforms:instance>
        <example xmlns="">
          <text/>
          <int/>
          <bool/>
          <date/>
          <time/>
          <datetime/>
        </example>
```

Инстанс модели данных содержит несколько элементов, соответствующих различным типам данных.

```
</xforms:instance>

<!-- Элементы bind указывают тип данных -->
<!-- Тип данных указывается не для элемента управления, а для
элемента данных в модели -->
<xforms:bind nodeset="bool" type="xforms:boolean"/>
<xforms:bind nodeset="int" type="xforms:integer"/>
<xforms:bind nodeset="date" type="xforms:date"/>
```

```
<xforms:bind nodeset="time" type="xforms:time"/>
<xforms:bind nodeset="datetime" type="xforms:dateTime"/>
```

Элемент «xforms:bind» содержит обязательный атрибут «nodeset», в котором указывается XPath-выражение для связи с XML-элементом в инстансе.

В атрибуте «type» элемента «xforms:bind» задается тип данных элемента.

```
<!-- submission указывает куда необходимо отправить форму -->
<xforms:submission action="" method="post" id="submit_id"
includenamespaceprefixes=""/>
</xforms:model>
</head>
```

```
<body>
<xforms:group>
  <xforms:input ref="text">
    <xforms:label>Введите текст:</xforms:label>
  </xforms:input>
```

Поле ввода данных не содержит информацию о типе данных элемента. Информация о типе данных автоматически берется из модели.

```
<xforms:input ref="int">
  <xforms:label>Введите число:</xforms:label>
  <xforms:alert>Значение должно быть целым
числом</xforms:alert>
```

Элемент «xforms:alert» содержит сообщение об ошибке, которое выводится в случае ошибки.

```
</xforms:input>

<xforms:input ref="bool">
  <xforms:label>Введите логическое значение:</xforms:label>
</xforms:input>

<xforms:input ref="date">
  <xforms:label>Введите дату:</xforms:label>
  <xforms:alert>Значение должно быть датой</xforms:alert>
</xforms:input>
```

```

<xforms:input ref="time">
  <xforms:label>Введите время:</xforms:label>
  <xforms:alert>Значение должно быть временем</xforms:alert>
</xforms:input>

<xforms:input ref="datetime">
  <xforms:label>Введите дату и время:</xforms:label>
</xforms:input>

<!-- Кнопка отправки формы. Атрибут submission ссылается на
элемент xforms:submission в модели -->
<xforms:submit submission="submit_id">
  <xforms:label>Отправка формы</xforms:label>
</xforms:submit>

</xforms:group>
</body>
</html>

```

Результат выполнения примера (ошибка ввода числа):

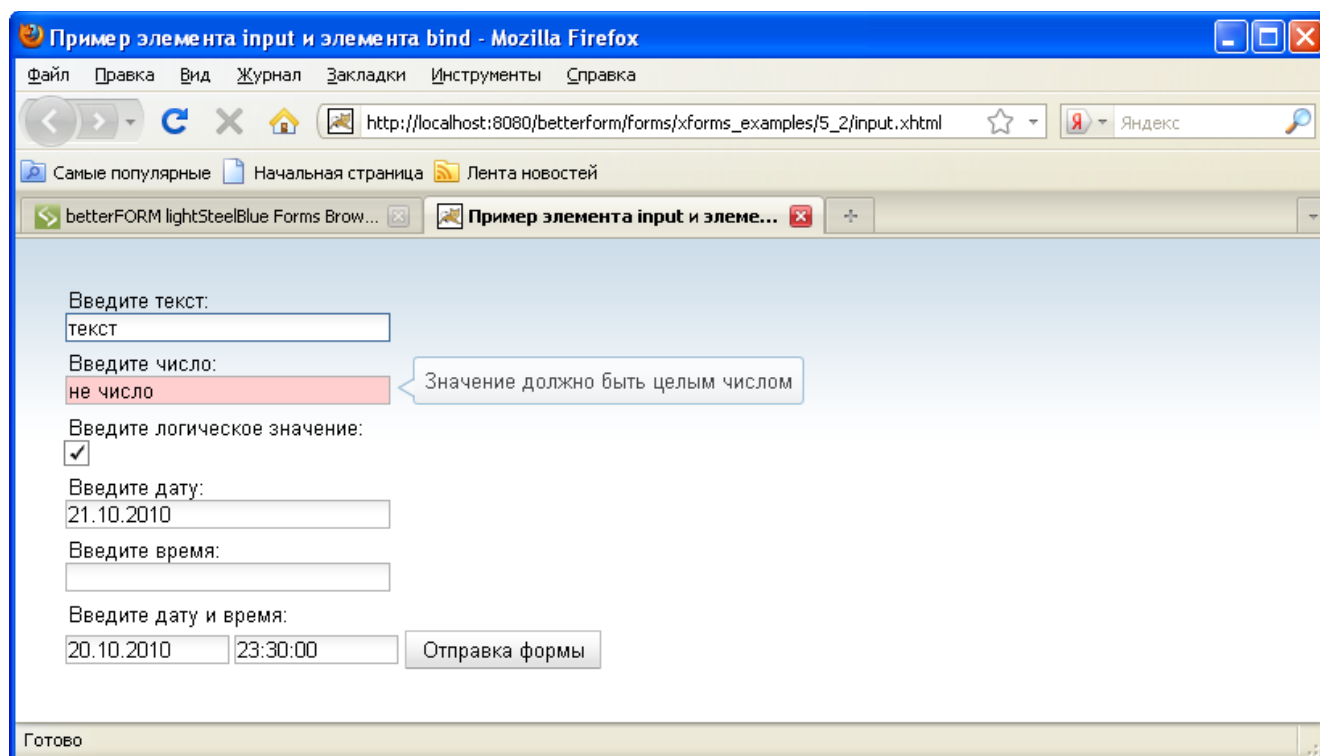


Рис. 5.4. Результат выполнения примера 5.2. Файл «input.xhtml».

Как правило, большинство XForms-процессоров модифицируют элемент «xforms:input» в зависимости от типа данных элемента. Если элемент логического типа, то он отображается в виде флажка, если типа «дата», то появляется всплывающее окно с выбором даты и т.д.

### 5.6.2 Связь данных формы, ограничений и элементов управления формы

В предыдущем примере использовался вариант связи с использованием атрибута «ref».

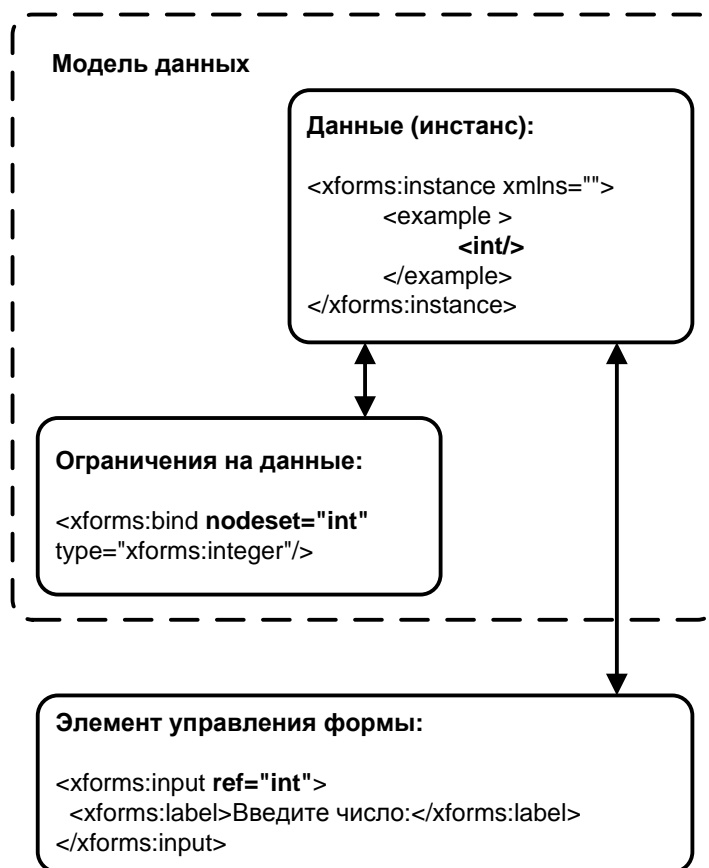


Рис. 5.5. Вариант связи данных формы, ограничений и элементов управления с использованием атрибута «ref».

Этот вариант является рекомендуемым и используется в большинстве случаев.

Элемент управления формы связан напрямую с экземпляром данных с помощью атрибута «ref=XPath-выражение». XPath-выражение выбирает необходимый элемент из экземпляра.

В случае необходимости в модель добавляются ограничения на данные с помощью элементов «xforms:bind».

Элемент «xforms:bind» также связан напрямую с экземпляром данных с помощью атрибута «nodeset=XPath-выражение».

Как правило, для одного элемента данных атрибуты «ref» и «nodeset» содержат одинаковые XPath-выражения.

Этот вариант связи удобен тем, что ограничения на данные никак не связаны с элементами управления. Ограничения можно добавлять и удалять по мере необходимости. При этом ограничения учитываются XForms-процессором при вводе данных. Например, в случае логического типа данных, элемент «xforms:input» отображается в виде флажка.

Но почему элемент для задания ограничений назван «xforms:bind»? Ведь в переводе «bind» означает связь или связывание, а не ограничение.

В ранних версиях спецификации XForms вариант связи с использованием атрибута «ref» считался дополнительным. А рекомендуемым был вариант связи с использованием атрибута «bind».

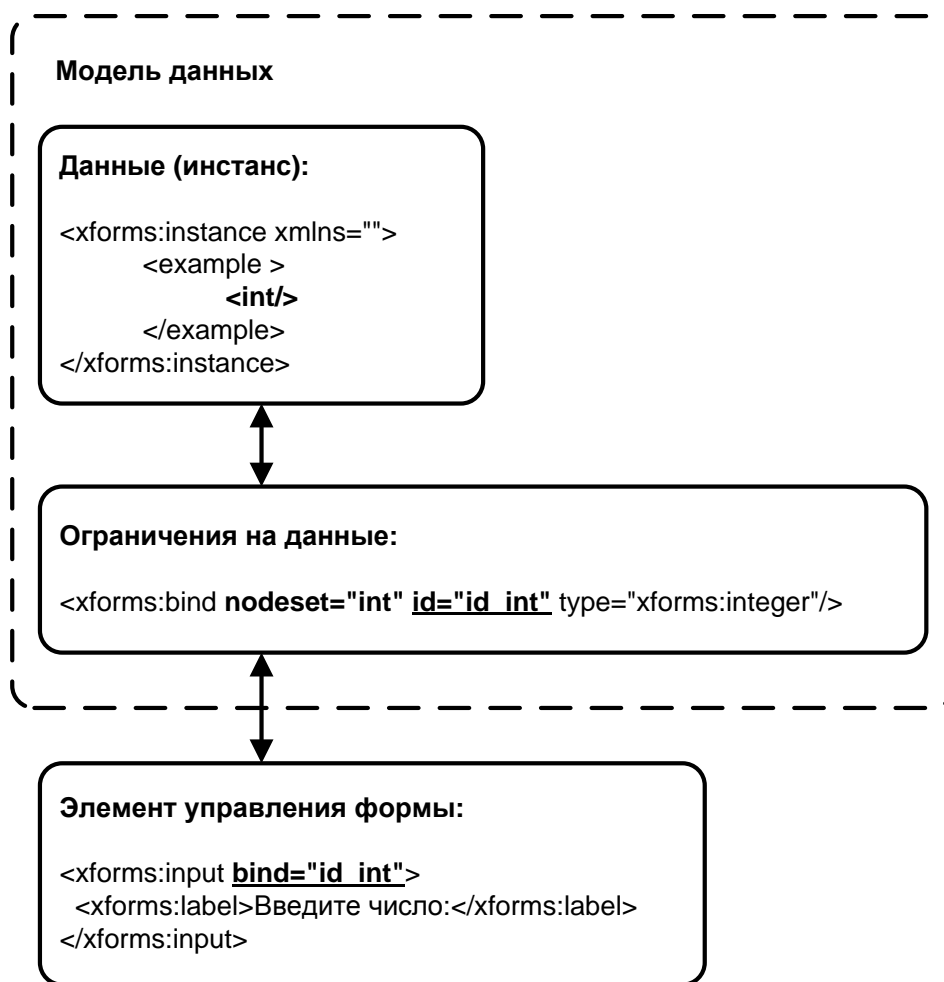


Рис. 5.6. Вариант связи данных формы, ограничений и элементов управления с использованием атрибута «bind».

В этом варианте элемент «xforms:bind» связан напрямую с инстансом данных с помощью атрибута «nodeset=XPath-выражение». Также у элемента «xforms:bind» должен быть задан атрибут «id».

Элемент управления формы связан с элементом «xforms:bind» с помощью атрибута «bind=id элемента xforms:bind».

Недостатком этого варианта является то, что в модели всегда необходимо создавать элемент «xforms:bind», даже если у XML-данных нет ограничений.

Поэтому сейчас, как правило, используется предыдущий вариант с атрибутом «ref».

Однако, вариант с атрибутом «bind» поддерживается большинством XForms-процессоров. Следующий пример использует вариант с атрибутом «bind».



### Пример 5.2. Файл «input\_bind.xhtml».

```

<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример элемента input (связь через bind)</title>

    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model>
      <xforms:instance>
        <example xmlns="">
          <text/>
          <int/>
          <bool/>
          <date/>
          <time/>
          <datetime/>
        </example>
      </xforms:instance>

      <!-- Элементы bind указывают тип данных -->
      <!-- Тип данных указывается не для элемента управления, а для
элемента данных в модели -->
      <xforms:bind id="id bool" nodeset="bool"
type="xforms:boolean"/>
      <xforms:bind id="id int" nodeset="int" type="xforms:integer"/>
      <xforms:bind id="id date" nodeset="date" type="xforms:date"/>
      <xforms:bind id="id time" nodeset="time" type="xforms:time"/>
      <xforms:bind id="id datetime" nodeset="datetime"
type="xforms:dateTime"/>

      <!-- submission указывает куда необходимо отправить форму -->

```

```

    <xforms:submission action="" method="post" id="submit_id"
includenamespaceprefixes=""/>
  </xforms:model>
</head>

```

```

<body>

```

```

  <xforms:group>

```

```

    <xforms:input ref="text">

```

```

      <xforms:label>Введите текст:</xforms:label>

```

```

    </xforms:input>

```

<!-- Вместо атрибута ref="int" используется атрибут bind="id\_int", то есть элемент пользовательского интерфейса привязывается не к элементу данных модели, а к элементу bind. Такой вариант допустим, но рекомендуется использовать атрибут ref -->

```

    <xforms:input bind="id_int">

```

```

      <xforms:label>Введите число:</xforms:label>

```

```

      <xforms:alert>Значение должно быть целым

```

```

числом</xforms:alert>

```

```

    </xforms:input>

```

```

    <xforms:input bind="id_bool">

```

```

      <xforms:label>Введите логическое значение:</xforms:label>

```

```

    </xforms:input>

```

```

    <xforms:input ref="date">

```

```

      <xforms:label>Введите дату:</xforms:label>

```

```

    </xforms:input>

```

```

    <xforms:input bind="id_time">

```

```

      <xforms:label>Введите время:</xforms:label>

```

```

    </xforms:input>

```

```

    <xforms:input bind="id_datetime">

```

```

      <xforms:label>Введите дату и время:</xforms:label>

```

```

</xforms:input>

<!-- Кнопка отправки формы. Атрибут submission ссылается на
элемент xforms:submission в модели -->
<xforms:submit submission="submit_id">
  <xforms:label>Отправка формы</xforms:label>
</xforms:submit>

</xforms:group>
</body>
</html>

```

Результат выполнения такой же, как в предыдущем примере.

### 5.6.3 Типы данных в XForms

Для проверки правильности ввода данных XForms поддерживает большинство типов данных XML Schema.

В спецификации XForms отмечается, что не поддерживаются `xsd:duration`, `xsd:ENTITY`, `xsd:ENTITIES`, и `xsd:NOTATION`.

Все эти типы определены в пространстве имен XML-схем, и для них используется префикс `xsd`.

Но с точки зрения XForms, у большинства этих типов есть существенный недостаток – пустое значение в них считается «неправильным». Однако, в XForms пустое значение должно считаться «правильным», так как пользователь может не вводить данные.

Для решения этой проблемы были введены типы данных в пространстве имен XForms. В отличие от типов данных XML-схем пустое значение в них считается «правильным». Названия этих типов данных совпадают с названиями типов данных в XML-схемах, но в качестве префикса пространства имен используется префикс XForms.

Поэтому в предыдущих примерах использовались типы «`xforms:integer`», «`xforms:date`», а не «`xsd:integer`», «`xsd:date`».

Полный список дополнительных типов в пространстве имен XForms приведен в главе 5 (Datatypes) спецификации XForms [XForms, 2009].

В следующем списке перечислены наиболее часто используемые типы в пространстве имен XForms:

- xforms:dateTime
- xforms:time
- xforms:date
- xforms:string
- xforms:boolean
- xforms:float
- xforms:decimal
- xforms:double
- xforms:anyURI
- xforms:QName
- xforms:integer
- xforms:nonPositiveInteger
- xforms:negativeInteger
- xforms:long
- xforms:int

Значения этих типов данных рассмотрены в спецификации XML-схем [XML Schema Part 2 Datatypes, 2004].

Отличие от соответствующих типов в XML-схеме состоит в том, что в XML-схеме «пустое» значение данных для большинства типов считается «неправильным» значением, а в пространстве имен XForms пустое значение допустимо.

Интересной особенностью XForms является возможность объявлять собственные типы на основе типов XML-схем и использовать их при проверке данных.

Создаваемый тип данных должен быть простым (xsd:simpleType), он соответствует одному XML-элементу (одному полю ввода), и может использовать фасеты XML-схем для задания дополнительных ограничений.

Составной тип данных (xsd:complexType) в этом случае использоваться не может, так как он объявляет тип данных из нескольких XML-элементов, что соответствовало бы нескольким полям ввода.

В следующем примере создается новый тип данных на основе перечисления (xsd:enumeration) и этот тип используется при проверке данных.

### **Пример 5.2. Файл «input\_newtype.xhtml».**

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

В пространствах имен дополнительно объявляется пространство имен для XML-схемы.

```
<head>
<title>Пример элемента bind</title>
```

```
<!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
<xforms:model>
```

```
<!-- Создание нового типа данных, который используется при
проверке. Тип данных объявляется в модели. -->
```

```
<xsd:schema>
  <xsd:simpleType name="NewType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value='abc' />
      <xsd:enumeration value='123' />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Создание нового типа данных NewType. Это строка, которая может содержать значение «abc» или «123».

Объявление нового типа данных (элемент «xsd:schema») вложено в модель данных (элемент «xforms:model»).

```
<xforms:instance>
  <example xmlns="">
    <newtype/>
  </example>
</xforms:instance>
```

<!-- Тип данных указывается не для элемента управления, а для элемента данных в модели -->

```
<xforms:bind nodeset="newtype" type="NewType"/>
```

<!-- submission указывает куда необходимо отправить форму -->

```
<xforms:submission action="" method="post" id="submit_id"
includenamespaceprefixes=""/>
```

```
</xforms:model>
```

```
</head>
```

```
<body>
```

```
<xforms:group>
```

```
<xforms:input ref="newtype">
```

```
<xforms:label>Введите 'abc' или '123':</xforms:label>
```

```
<xforms:message ev:event="xforms-valid">Введено правильное
значение</xforms:message>
```

```
<xforms:message ev:event="xforms-invalid">Введено
НЕправильное значение</xforms:message>
```

```
</xforms:input>
```

Элемент «xforms:message» выводит всплывающее окно с сообщением. Здесь этот элемент используется как обработчик событий «xforms-valid» (форма заполнена правильно) и «xforms-invalid» (форма заполнена неправильно).

<!-- Кнопка отправки формы. Атрибут submission ссылается на элемент xforms:submission в модели -->

```
<xforms:submit submission="submit_id">
```

```
<xforms:label>Отправка формы</xforms:label>
</xforms:submit>
```

```
</xforms:group>
```

```
</body>
```

```
</html>
```

Результат выполнения примера (ввод правильного и неправильного значения):

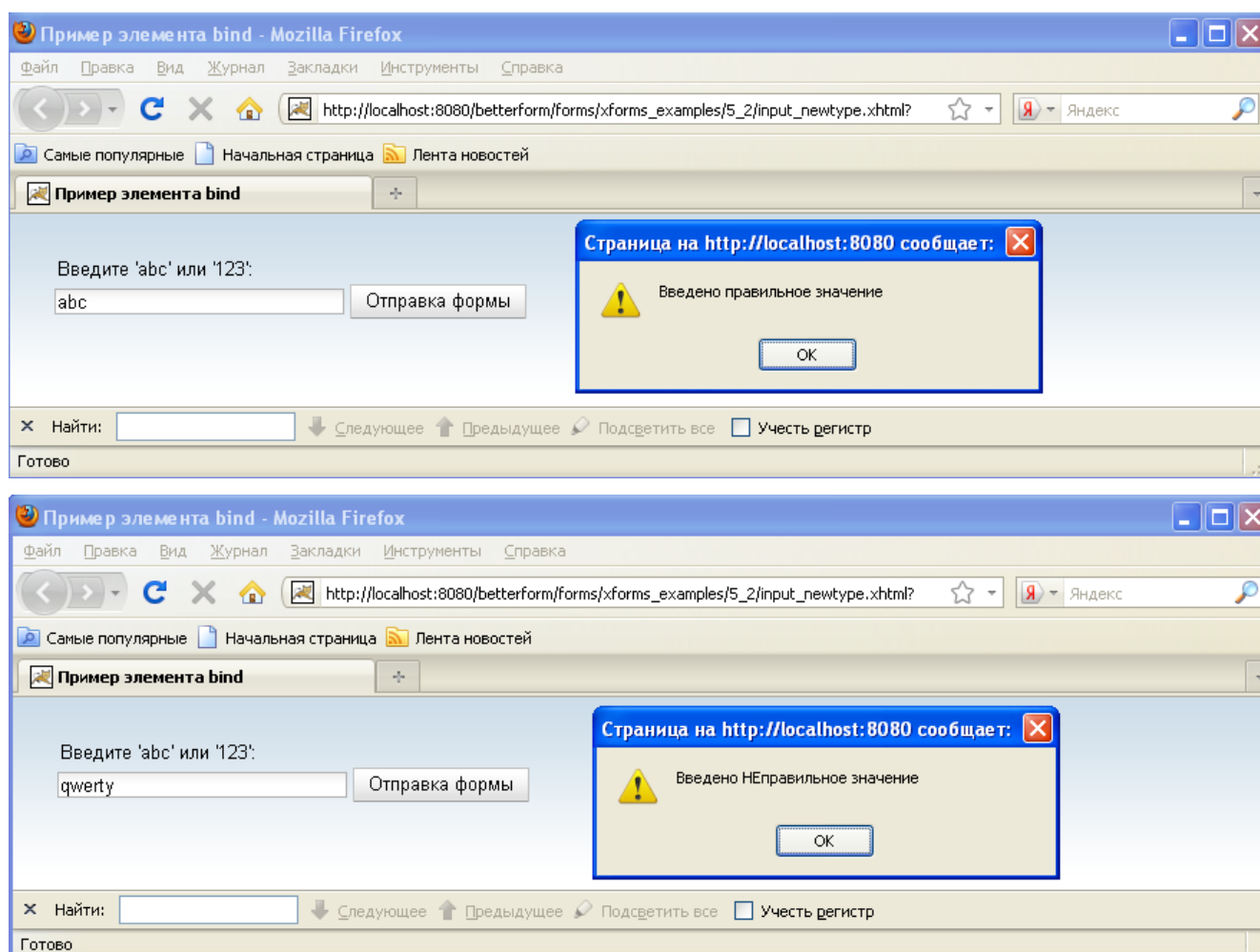


Рис. 5.7. Результат выполнения примера 5.2. Файл «input\_newtype.xhtml».

В этом примере впервые используется реакция на события (атрибут «ev:event»).

Элемент «xforms:message» без атрибута «ev:event» всегда выводит заданное сообщение. Но с атрибутом «ev:event=название события» он превращается в обработчик события, сообщение выводится, только если происходит событие.

#### **5.6.4 Задание ограничений на данные с помощью элемента xforms:bind**

В предыдущих примерах мы задавали только ограничения на типы данных. Но с помощью элемента xforms:bind можно задавать различные ограничения.

Несмотря на то, что ограничения накладываются на данные в модели данных, пользователь фактически сталкивается с этими ограничениями в элементах управления формы. Например, если на элемент данных наложено ограничение «только для чтения», то соответствующее поле ввода не позволяет вводить данные.

В следующей таблице приведены наиболее часто используемые атрибуты элемента xforms:bind.



Таблица 5.1. Атрибуты элемента xforms:bind.

Атрибут	Значение
nodeset="XPath-выражение, задающее ссылку на элемент данных"	Привязка элемента xforms:bind к элементу данных
type="тип данных"	Задание ограничения на тип данных
readonly="XPath-выражение, возвращающее логическое значение"	Элемент данных доступен только для чтения
required="XPath-выражение, возвращающее логическое значение"	Элемент данных не может быть пустым, то есть в соответствующий полю элемент управления должны быть введены данные
relevant="XPath-выражение, возвращающее логическое значение"	Соответствующий элемент управления отображается для ввода данных (если значение ложно, то не отображается)
calculate="XPath-выражение для вычисления значения элемента"	Используется для создания вычисляемых полей. Значение XPath-выражения вычисляется и сохраняется как данные XML-элемента. Часто этот атрибут используется вместе с атрибутом readonly.
constraint="XPath-выражение, возвращающее логическое значение"	XML-элемент содержит правильные данные, если выражение истинно

В следующем примере используются различные типы ограничений.

**Пример 5.2. Файл «binds.xhtml».**

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
<head>
```

```
<title>Пример элемента bind</title>
```

```
<!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
```

```
<xforms:model>
```

```
<!-- Создание нового типа данных, который используется при
проверке -->
```

```
<xsd:schema>
```

```
<xsd:simpleType name="NewType">
```

```
<xsd:restriction base="xsd:string">
```

```
<xsd:enumeration value='abc' />
```

```
<xsd:enumeration value='123' />
```

```
</xsd:restriction>
```

```
</xsd:simpleType>
```

```
</xsd:schema>
```

```
<xforms:instance>
```

```
<example xmlns="">
```

```
<flag>1</flag>
```

```
<bool>true</bool>
```

```
<date/>
```

```
<newtype/>
```

```
<readonlytext/>
```

```
<requiredtext/>
```

```
<reltext/>
```

```
<number1>200</number1>
```

[Оглавление](#)

```

    <number2>100</number2>
    <number_percent/>
    <number_constraint>150</number_constraint>
  </example>
</xforms:instance>

```

**<!-- Ограничения на тип данных (стандартный или созданный с помощью XML-схемы) -->**

```
<xforms:bind nodeset="date" type="xforms:date"/>
```

```
<xforms:bind nodeset="newtype" type="NewType"/>
```

**<!-- required="true()" означает, что поле обязательно для ввода -->**

```
<xforms:bind nodeset="requiredtext" required="true()"/>
```

**<!-- readonly="/example/flag!=1" означает, что поле недоступно для ввода данных (доступно только для чтения) если элемент flag (вложенный в элемент example) не равен 1 -->**

```
<xforms:bind nodeset="readonlytext" readonly="/example/flag!=1"/>
```

**<!-- relevant="/example/flag=1" означает, что поле отображается в форме если элемент flag (вложенный в элемент example) равен 1 -->**

```
<xforms:bind nodeset="reltext" relevant="/example/flag=1"/>
```

```
<xforms:bind nodeset="number1" type="xforms:float"/>
```

```
<xforms:bind nodeset="number2" type="xforms:float"/>
```

```

<xforms:bind nodeset="number_percent" readonly="true()"
type="xforms:float" relevant="(/example/number1 > 0) and
(/example/number2 > 0)" calculate="round(((/example/number2) div
(/example/number1))*100)"/>

```

Элемент "number\_percent" доступен только для чтения (readonly="true()").

Атрибут relevant="(/example/number1 > 0) and (/example/number2 > 0)" означает, что поле отображается, только если элементы example/number1 и example/number2 больше нуля.

Атрибут `calculate="round((((/example/number2) div (/example/number1))*100))"` вычисляет процентное значение.

Значение автоматически пересчитывается при изменении зависимых XML-элементов (в нашем случае `number1` и `number2`).

Вычисленное значение сохраняется в элемент `"number_percent"`.

```
<xforms:bind nodeset="number_constraint" type="xforms:float"
constraint="(. &lt;= /example/number1) and (. &gt;=
/example/number2)" />
```

Атрибут `constraint="(. &lt;= /example/number1) and (. &gt;= /example/number2)"` означает, что поле заполнено правильно, если значение XPath-выражения в атрибуте `constraint` истинно.

В нашем случае значение элемента, для которого задается ограничение `"number_constraint" (.)`, должно быть меньше или равно элементу `example/number1 (. &lt;= /example/number1)` и должно быть больше или равно элементу `example/number2 (. &gt;= /example/number2)`.

```
<!-- submission указывает, куда необходимо отправить форму -->
<xforms:submission action="" method="post" id="submit_id"
includenamespacesprefixes="" />
</xforms:model>
</head>

<body>
  <xforms:group>

    <xforms:input ref="date">
      <xforms:label>Введите дату:</xforms:label>
    </xforms:input>

    <xforms:input ref="newtype">
      <xforms:label>Введите 'abc' или '123':</xforms:label>
```

```

    <xforms:message ev:event="xforms-valid">Введено правильное
значение</xforms:message>
    <xforms:message ev:event="xforms-invalid">Введено
НЕправильное значение</xforms:message>
  </xforms:input>

  <xforms:input ref="requiredtext">
    <xforms:label>Это поле обязательно для ввода:</xforms:label>
  </xforms:input>

  <xforms:input ref="readonlytext">
    <xforms:label>Это поле доступно для ввода данных только
когда флаг установлен в 1:</xforms:label>
  </xforms:input>

  <xforms:input ref="reltext">
    <xforms:label>Это поле отображается для ввода данных только
когда флаг установлен в 1:</xforms:label>
  </xforms:input>

```

#### **<xforms:trigger>**

Элемент «xforms:trigger» создает кнопку.

```

  <xforms:label>Установить флаг</xforms:label>
  <xforms:setvalue ref="flag"
ev:event="DOMActivate">1</xforms:setvalue>

```

Элемент «xforms:setvalue» в данном случае является обработчиком события. Атрибут `ev:event="DOMActivate"` означает, что элемент `xforms:setvalue` должен реагировать на «стандартное» событие элемента «xforms:trigger», для «xforms:trigger» таким стандартным событием является нажатие кнопки.

Элемент «xforms:setvalue» устанавливает значение элемента данных `flag` в 1.

#### **</xforms:trigger>**

```

<xforms:trigger>
  <xforms:label>Сбросить флаг</xforms:label>

```

```

    <xforms:setvalue ref="flag"
ev:event="DOMActivate">0</xforms:setvalue>
    </xforms:trigger>

    <xforms:output value="concat('Значение флага: ', flag)"/>

    <xforms:input ref="/example/number1">
        <xforms:label>Число 1:</xforms:label>
    </xforms:input>

    <xforms:input ref="/example/number2">
        <xforms:label>Число 2:</xforms:label>
    </xforms:input>

    <xforms:input ref="/example/number_percent">
        <xforms:label>Число 2 составляет от числа 1
процентов:</xforms:label>
    </xforms:input>

    <xforms:input ref="/example/number_constraint">
        <xforms:label>Введите число в диапазоне от число 1 до число
2:</xforms:label>
        <xforms:message ev:event="xforms-valid">Введено правильное
значение в диапазоне от число 1 до число 2</xforms:message>
        <xforms:message ev:event="xforms-invalid">Введено
НЕправильное значение в диапазоне от число 1 до число
2</xforms:message>
    </xforms:input>

    <!-- Кнопка отправки формы. Атрибут submission ссылается на
элемент xforms:submission в модели -->
    <xforms:submit submission="submit_id">
        <xforms:label>Отправка формы</xforms:label>
    </xforms:submit>

```

```

</xforms:group>
</body>
</html>

```

Результат выполнения примера:

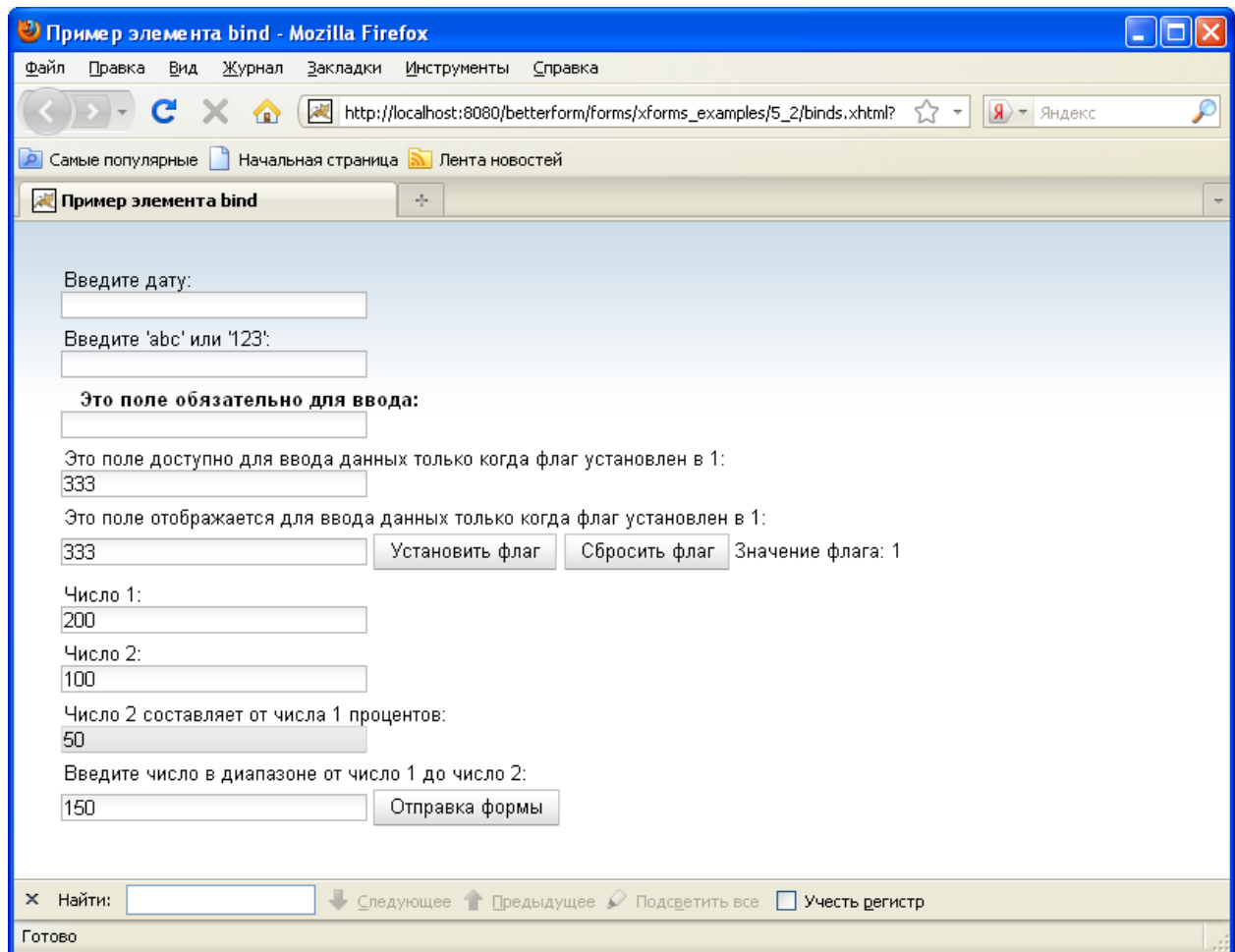


Рис. 5.8. Результат выполнения примера 5.2. Файл «binds.xhtml».

### 5.6.5 Использование нескольких моделей и инстансов

В предыдущих примерах использовались одна модель данных и один инстанс данных в модели.

Однако, в XForms возможно использование нескольких моделей данных и нескольких инстансов в каждой модели.

В случае использования нескольких моделей данных в элементах управления формы необходимо вместе с атрибутом «ref» использовать атрибут «model=id модели» для ссылки на модель.

Если вместо атрибута «ref» используется атрибут «bind=id элемента xforms:bind», то ссылку на модель использовать не нужно, так как ссылка на id элемента xforms:bind уникальна в рамках документа и автоматически определяет модель данных.

По умолчанию ссылки на данные соответствуют первому экземпляру в модели.

Если модель содержит несколько экземпляров, то у этих экземпляров необходимо задавать атрибут «id», и в XPath-выражениях для ссылки на эти экземпляры использовать функцию «instance('id экземпляра')».

Для доступа к элементам данных можно использовать XPath-выражение вида «instance('id экземпляра')//элемент данных». Такие ссылки на XML-данные необходимо использовать и в элементах управления формы (атрибут «ref»), и в элементах xforms:bind (атрибут «nodeset»).

В следующем примере показано, как в случае использования нескольких моделей и экземпляров может осуществляться привязка данных.

### **Пример 5.2. Файл «models.xhtml».**

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример использования нескольких моделей</title>

    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model id="model1">

      <xforms:instance>
        <example xmlns="">
          <text/>
```



```

    </example>
</xforms:instance>

```

```

<xforms:instance id="instance_2">
    <example xmlns="">
        <bool/>
    </example>
</xforms:instance>

```

```

<!-- Элементы bind указывают тип данных -->
<!-- Тип данных указывается не для элемента управления, а для
элемента данных в модели -->
    <xforms:bind nodeset="instance('instance_2')//bool"
type="xforms:boolean"/>

<!-- submission указывает куда необходимо отправить форму -->
    <xforms:submission action="" method="post" id="submit_id1"
includenamespaceprefixes=""/>
</xforms:model>

```

```

<xforms:model id="model12">
    <xforms:instance>
        <example xmlns="">
            <date/>
            <datetime/>
        </example>
    </xforms:instance>

```

```

<!-- Элементы bind указывают тип данных -->
<!-- Тип данных указывается не для элемента управления, а для
элемента данных в модели -->
    <xforms:bind nodeset="date" type="xforms:date"/>
    <xforms:bind id="id_datetime" nodeset="datetime"
type="xforms:dateTime"/>

```

```

        <!-- submission указывает куда необходимо отправить форму -->
        <xforms:submission action="" method="post" id="submit_id2"
includenamespacesprefixes=""/>
    </xforms:model>

</head>

<body>
    <xforms:group>
        <!-- Если в форме задано несколько моделей, то кроме элемента
ref должен использоваться атрибут model для ссылки на модель -->
        <xforms:input ref="text" model="model1">
            <xforms:label>Введите текст:</xforms:label>
        </xforms:input>

        <xforms:input ref="instance('instance_2')//bool"
model="model1">
            <xforms:label>Введите логическое значение:</xforms:label>
        </xforms:input>

        <xforms:submit submission="submit_id1">
            <xforms:label>Отправка формы 1</xforms:label>
        </xforms:submit>

        <xforms:input ref="date" model="model2">
            <xforms:label>Введите дату:</xforms:label>
        </xforms:input>

        <xforms:input bind="id_datetime">
            <xforms:label>Введите дату и время:</xforms:label>
        </xforms:input>

        <xforms:submit submission="submit_id2">
            <xforms:label>Отправка формы 2</xforms:label>
        </xforms:submit>

```

```

    </xforms:group>
</body>
</html>

```

Результат выполнения примера:

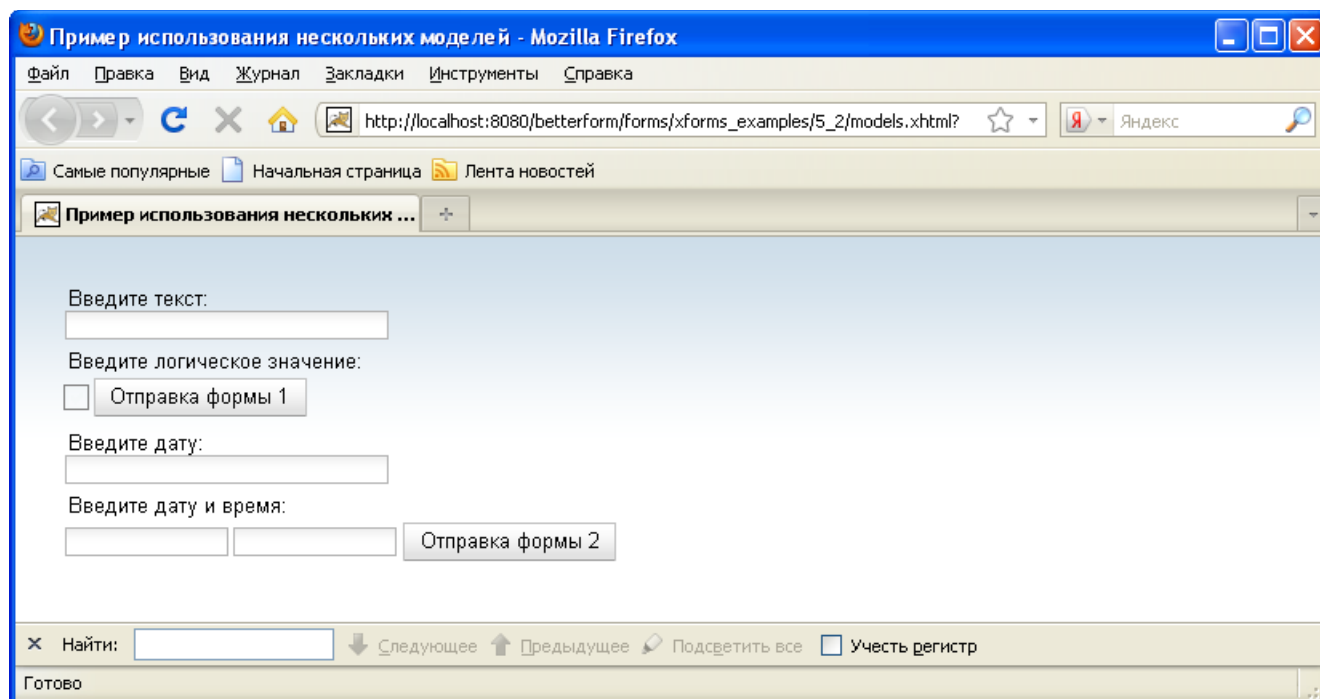


Рис. 5.9. Результат выполнения примера 5.2. Файл «models.xhtml».

В предыдущих примерах данные по-умолчанию всегда задавались как содержимое элемента `xforms:instance`. Но в XForms данные по-умолчанию можно задать тремя способами:

1. Как содержимое элемента `xforms:instance`.
2. Через атрибут `src`, в котором указывается URI источника данных (например, URI серверного сценария, который формирует данные).
3. Через атрибут `resource`, в котором указывается URI источника данных.

При этом используются следующие правила:

1. Если указан атрибут `src`, то он имеет более высокий приоритет по сравнению с содержимым элемента `xforms:instance` и с атрибутом `resource`.

2. Если атрибут `src` не указан, то содержимое элемента `xforms:instance` имеет больший приоритет по сравнению с атрибутом `resource`.

Таким образом, с использованием атрибутов «`src`» и «`resource`» элемента «`xforms:instance`» можно загружать данные по-умолчанию из дополнительных источников.

Проблема заключается в том, что в реальных приложениях значение атрибутов «`src`» и «`resource`» может меняться (например, в случае создания нового элемента данных и в случае редактирования ранее введенных данных). То есть желательно или параметризовать XForms-форму, или сгенерировать ее динамически.

Поэтому при разработке приложений с использованием XForms мы будем использовать XForms-процессор XSLTForms. XSLTForms позволяет достаточно просто работать с XForms-формами, которые сгенерированы динамически с помощью серверных XQuery- сценариев.

## **5.7 Элементы управления формы**

Элементы управления формы позволяют пользователю вводить данные в модель данных.

К элементам управления формы относятся поля ввода, радиокнопки, списки и т.д.

Некоторые из этих элементов управления имеют прямые аналоги в HTML-формах, некоторые являются более сложными элементами и не имеют прямых аналогов в HTML-формах.

Все элементы управления можно разделить на две группы:

1. Элементы управления, которые используются для ввода данных в единственный XML-элемент данных.

Это такие элементы управления, как:

- `input` (поле ввода).
- `secret` (поле ввода со скрытием символов).

- textarea (многострочное поле ввода).
- output (поле вывода).
- upload (поле выбора файла).
- range (выбор значения шкалы).
- label (задание подписи к элементу).
- help (справочная информация об элементе).
- hint (короткая справка об элементе).
- alert (сообщение об ошибке).
- trigger (кнопка или гиперссылка).
- submit (кнопка или гиперссылка отправки данных формы на сервер).
- select (список с множественным выбором).
- select1 (список с единичным выбором).

Для связи таких элементов управления с XML-данными используется способ, который в спецификации называется «Single-Node Binding».

Этот способ мы уже рассмотрели ранее. Он предполагает два способа для ссылки на элемент данных:

I. Использование атрибута «ref=XPath-выражение» и, при необходимости ссылки на модель, атрибута «model=id модели».

II. Использование атрибута «bind=id элемента xforms:bind». В этом случае id элемента xforms:bind однозначно определяет модель.

При этом предполагается, что мы выбираем из модели данных единственный элемент.

2. Элементы управления, которые используются для обработки группы XML-элементов данных.

Это такие элементы управления, как:

- group (группировка элементов).
- switch (создание и переключение вкладок).
- repeat (обработка повторяющиеся данных).

Для связи таких элементов управления с XML-данными используется способ, который в спецификации называется «Node-Set Binding».

Он также предполагает два способа для ссылки на элементы данных:

I. Использование атрибута «nodeset=XPath-выражение» и, при необходимости ссылки на модель, атрибута «model=id модели».

II. Использование атрибута «bind=id элемента xforms:bind». В этом случае id элемента xforms:bind однозначно определяет модель.

При этом предполагается, что мы выбираем из модели данных группу элементов.

Большинство элементов управления позволяет использование следующих атрибутов.

Таблица 5.2. Атрибуты элементов управления.

Атрибут	Значение
appearance = "full"   "compact"   "minimal"	Отображает элемент в полном, компактном или минимальном виде. Как правило, этот атрибут влияет на большинство элементов управления, особенно на элементы отображения группы данных (repeat и т.д.).
mediatype="значение"	Позволяет указывать тип содержимого для данных, что меняет внешний вид элемента управления. Например, элемент textarea с атрибутом mediatype="text/html" некоторыми XForms-процессорами отображается не как многострочный текст, а как HTML-редактор.
navindex="целое число в диапазоне 0-32767"	Определяет порядок перехода по клавише «tab» между полями. Переход осуществляется к элементу с минимальным большим значением атрибута «navindex».
accesskey="горячая клавиша"	Позволяет задавать горячую клавишу для быстрого доступа к элементу управления. Как правило, используется комбинация alt+горячая клавиша.

Рассмотрим более подробно элементы управления, которые применяются в XForms.

### 5.7.1 Элемент input

Большинством XForms-процессоров отображается как поле ввода. Как правило, адаптируется к типу данных элемента. Например, в случае логического типа отображается в виде флажка.

Этот элемент активно использовался в предыдущих примерах.

### 5.7.2 Элемент secret

Большинством XForms-процессоров отображается как поле ввода со скрыванием вводимых символов. Обычно используется для ввода паролей.

#### Пример 5.3. Файл «secret.xhtml».

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример элемента secret</title>

    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model>
      <xforms:instance>
        <example xmlns="">
          <text/>
          <password/>
        </example>
      </xforms:instance>

      <!-- submission указывает, куда необходимо отправить форму -->
```

```

    <xforms:submission action="" method="post" id="submit_id"
includenamespaceprefixes=""/>
  </xforms:model>
</head>

<body>
  <xforms:group>
    <xforms:input ref="text">
      <xforms:label>Введите текст:</xforms:label>
    </xforms:input>

    <xforms:secret ref="password">
      <xforms:label>Введите пароль:</xforms:label>
    </xforms:secret>

    <!-- Кнопка отправки формы. Атрибут submission ссылается на
элемент xforms:submission в модели -->
    <xforms:submit submission="submit_id">
      <xforms:label>Отправка формы</xforms:label>
    </xforms:submit>

  </xforms:group>
</body>
</html>

```

Результат выполнения примера:

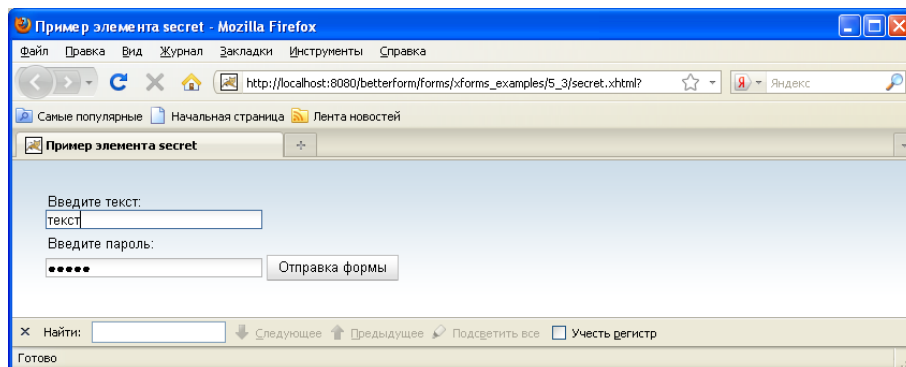


Рис. 5.10. Результат выполнения примера 5.3. Файл «secret.xhtml».



### 5.7.3 Элемент `textarea`

Большинством XForms-процессоров отображается как многострочное поле ввода.

Некоторые XForms-процессоры (в том числе betterFORM) в случае использования атрибута «`mediatype="text/html"`» отображают поле в виде редактора HTML.

Некоторые браузеры (например, Mozilla Firefox) автоматически проверяют орфографию в поле `textarea` (иногда также в поле `input`), точнее орфография автоматически проверяется в HTML-элементах, которые использует XForms-процессор.

Поскольку различные браузеры могут проверять орфографию для разных HTML-элементов и различные XForms-процессоры могут использовать разные HTML-элементы, то в каждом конкретном случае (браузер + XForms-процессор) требуются дополнительные исследования.

#### Пример 5.3. Файл «`textarea.xhtml`».

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример элемента textarea</title>

    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model>
      <xforms:instance>
        <example xmlns="">
          <text/>
          <textarea/>
        </example>
      </xforms:instance>
```

```

        <!-- submission указывает куда необходимо отправить форму -->
        <xforms:submission action="" method="post" id="submit_id"
includenamespaceprefixes=""/>
    </xforms:model>
</head>

<body>
    <xforms:group>
        <xforms:input ref="text">
            <xforms:label>Введите текст:</xforms:label>
        </xforms:input>

        <xforms:textarea ref="textarea">
            <xforms:label>Введите многострочный текст:</xforms:label>
        </xforms:textarea>

        <xforms:textarea ref="textarea" mediatype="text/html">
            <xforms:label>Введите многострочный текст
(HTML) :</xforms:label>
        </xforms:textarea>

        <!-- Кнопка отправки формы. Атрибут submission ссылается на
элемент xforms:submission в модели -->
        <xforms:submit submission="submit_id">
            <xforms:label>Отправка формы</xforms:label>
        </xforms:submit>

    </xforms:group>
</body>
</html>

```

Результат выполнения примера:

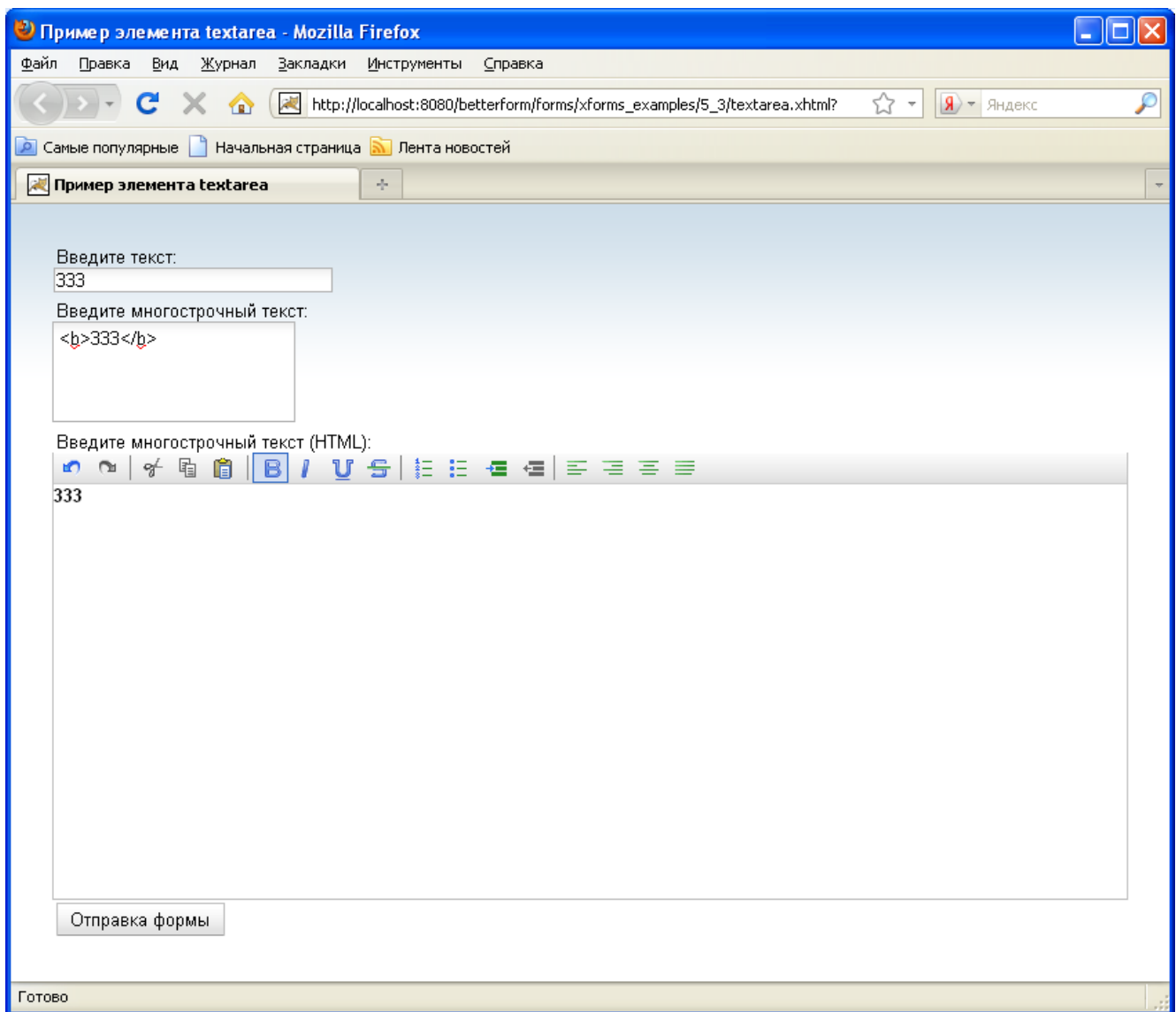


Рис. 5.11. Результат выполнения примера 5.3. Файл «textarea.xhtml».

#### 5.7.4 Элемент output

Большинством XForms-процессоров отображается как текст, в который нельзя вводить данные. Обычно используется для вывода вычисляемых полей.

Если в элементе «`xforms:output`» используется атрибут «`mediatype`», то с его помощью можно выводить данные различных типов (гиперссылки, изображения, фрагменты HTML и т.д.).

Вместо атрибута `mediatype` в элементе «`xforms:output`» может использоваться вложенный элемент `xforms:mediatype`.

#### Пример 5.3. Файл «output.xhtml».

[Оглавление](#)

```

<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <head>
    <title>Пример вывода</title>

    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model>
      <xforms:instance>
        <example xmlns="">
          <out1>Пример вывода 1</out1>
          <out2>http://localhost:8080/tomcat.gif</out2>
          <type_model>image/*</type_model>
          <html><![CDATA[ Разметка HTML <b>полужирный</b>
<i>курсив</i> <a href="http://localhost:8080/">главная
страница</a>]]></html>
        </example>
      </xforms:instance>

      <xforms:bind id="bind_out1" nodeset = "out1" />

      <!-- Элемент out2 имеет тип xsd:anyURI -->
      <xforms:bind nodeset = "out2" type="xsd:anyURI" />
      <!--

Кроме гиперссылок возможна ссылка на бинарные ресурсы типа
xsd:base64Binary или xsd:hexBinary, которые являются частью модели.

Бинарный ресурс должен быть декодирован и представлен,
например, в виде изображения
-->
    </xforms:model>
  </head>

```

```
<body>
```

```
<xforms:group>
```

```
<xforms:input ref="out1">
```

```
<xforms:label>Введите текст:</xforms:label>
```

```
</xforms:input>
```

```
<!-- Вывод текста из элемента. В атрибуте value используется
XPath-выражение, которое определяет выводимое значение -->
```

```
<!-- С использованием value элемент должен обновляться при
изменениях формы -->
```

```
<xforms:output value="concat('!!! ', out1, ' !!!')">
```

```
<xforms:label>Вывод текста с использованием
value:</xforms:label>
```

```
</xforms:output>
```

```
<!-- Возможно также выводить значения с использованием ref или
bind, но рекомендуется использовать value -->
```

```
<xforms:output ref="out1">
```

```
<xforms:label>Вывод текста с использованием
ref:</xforms:label>
```

```
</xforms:output>
```

```
<xforms:output bind="bind_out1">
```

```
<xforms:label>Вывод текста с использованием
bind:</xforms:label>
```

```
</xforms:output>
```

```
<!-- Если тип выводимого элемента xsd:anyURI, то он считается
гиперссылкой на ресурс -->
```

```
<!-- В случае использования атрибута ref гиперссылка выводится
как гиперссылка -->
```

```
<xforms:output ref="out2">
```

```
<xforms:label>Гиперссылка на изображение</xforms:label>
```

```
</xforms:output>
```

```
<!-- В случае использования атрибута value гиперссылка
выводится как текст -->
```

```
<xforms:output value="out2">
  <xforms:label>Вывод текста ссылки на
изображение:</xforms:label>
</xforms:output>
```

```
<!-- Атрибут mediatype указывает тип ресурса -->
<xforms:output value="out2" mediatype="image/*">
  <xforms:label>Вывод изображения (атрибут
mediatype):</xforms:label>
</xforms:output>
```

```
<!-- Тип ресурса может быть также указан с помощью элемента
mediatype и получен из модели с помощью атрибута value -->
<xforms:output value="out2">
  <xforms:mediatype>image/*</xforms:mediatype>
  <xforms:label>Вывод изображения (элемент
mediatype):</xforms:label>
</xforms:output>
```

```
<xforms:output value="out2">
  <xforms:mediatype value="type_model"/>
  <xforms:label>Вывод изображения (элемент mediatype со
ссылкой на модель):</xforms:label>
</xforms:output>
```

```
<!-- Атрибут mediatype указывает тип ресурса -->
<xforms:output value="html" mediatype="text/html">
  <xforms:label>Вывод фрагмента HTML:</xforms:label>
</xforms:output>
```

```
</xforms:group>
</body>
```

&lt;/html&gt;

Результат выполнения примера:

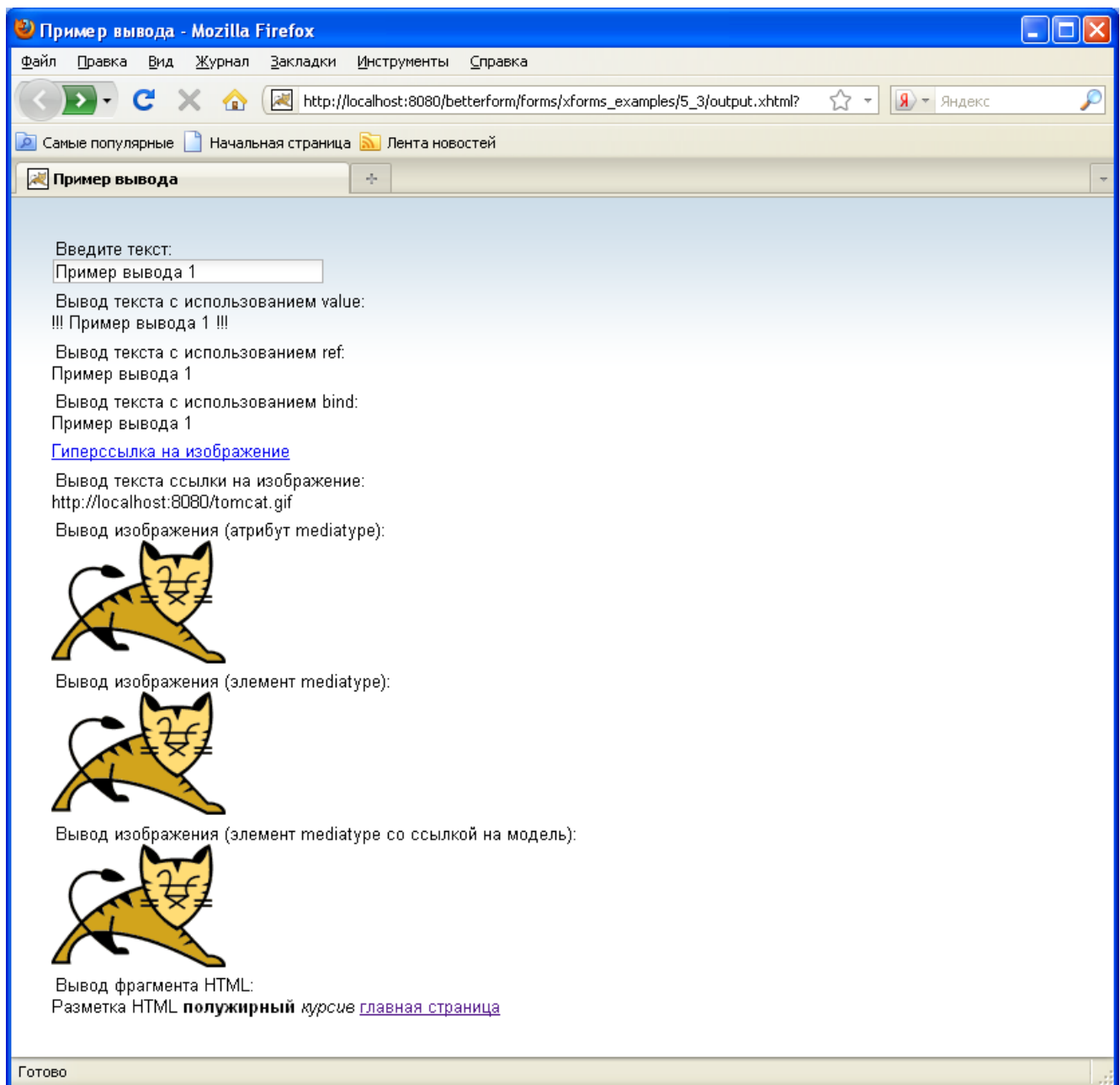


Рис. 5.12. Результат выполнения примера 5.3. Файл «output.xhtml».

### 5.7.5 Элемент upload

Большинством XForms-процессоров отображается как поле выбора файла.

Элемент данных, связанный с этим полем, может быть типа `xsd:anyURI`, `xsd:base64Binary` или `xsd:hexBinary`.

В случае типов данных `xsd:base64Binary` или `xsd:hexBinary` после выбора файла содержимое файла кодируется в соответствии с типом данных (`xsd:base64Binary` или `xsd:hexBinary`) и помещается в содержимое связанного элемента. То есть весь файл помещается в содержимое XML-элемента данных.

В случае типа данных `xsd:anyURI` в содержимое связанного элемента помещается только URI файла.

Однако, кроме содержимого файла, как правило, требуется сохранить его название и тип содержимого. В элемент «upload» могут быть вложены элементы «filename» и «mediatype» с атрибутом «ref». Атрибут «ref» указывает поля данных, в которые после выбора файла помещаются его название и тип содержимого.

### Пример 5.3. Файл «upload.xhtml».

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <head>
    <title>Пример загрузки</title>
```

```
<!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
```

```
  <xforms:model>
    <xforms:instance>
      <example xmlns="">
        <upload1/>
        <upload2/>
        <upload3/>
        <type/>
        <filename/>
      </example>
    </xforms:instance>
```



```

<xforms:bind nodeset = "upload1" type="xsd:anyURI" />
<xforms:bind nodeset = "upload2" type="xsd:base64Binary" />
<xforms:bind nodeset = "upload3" type="xsd:hexBinary" />

<xforms:submission action="" method="post" id="submit_id"
includenamespaceprefixes=""/>
</xforms:model>
</head>

<body>
<xforms:group>

<xforms:upload ref="upload1">
  <xforms:label>Загрузка URI:</xforms:label>
  <xforms:filename ref="filename"/>
  <xforms:mediatype ref="type"/>
</xforms:upload>

<xforms:upload ref="upload2">
  <xforms:label>Загрузка файла base64Binary:</xforms:label>
  <xforms:filename ref="filename"/>
  <xforms:mediatype ref="type"/>
</xforms:upload>

<xforms:upload ref="upload3">
  <xforms:label>Загрузка файла hexBinary:</xforms:label>
  <xforms:filename ref="filename"/>
  <xforms:mediatype ref="type"/>
</xforms:upload>

<!-- Кнопка отправки формы. Атрибут submission ссылается на
элемент xforms:submission в модели -->
<xforms:submit submission="submit_id">
  <xforms:label>Отправка формы</xforms:label>
</xforms:submit>

```

```

    </xforms:group>
  </body>
</html>

```

Результат выполнения примера:

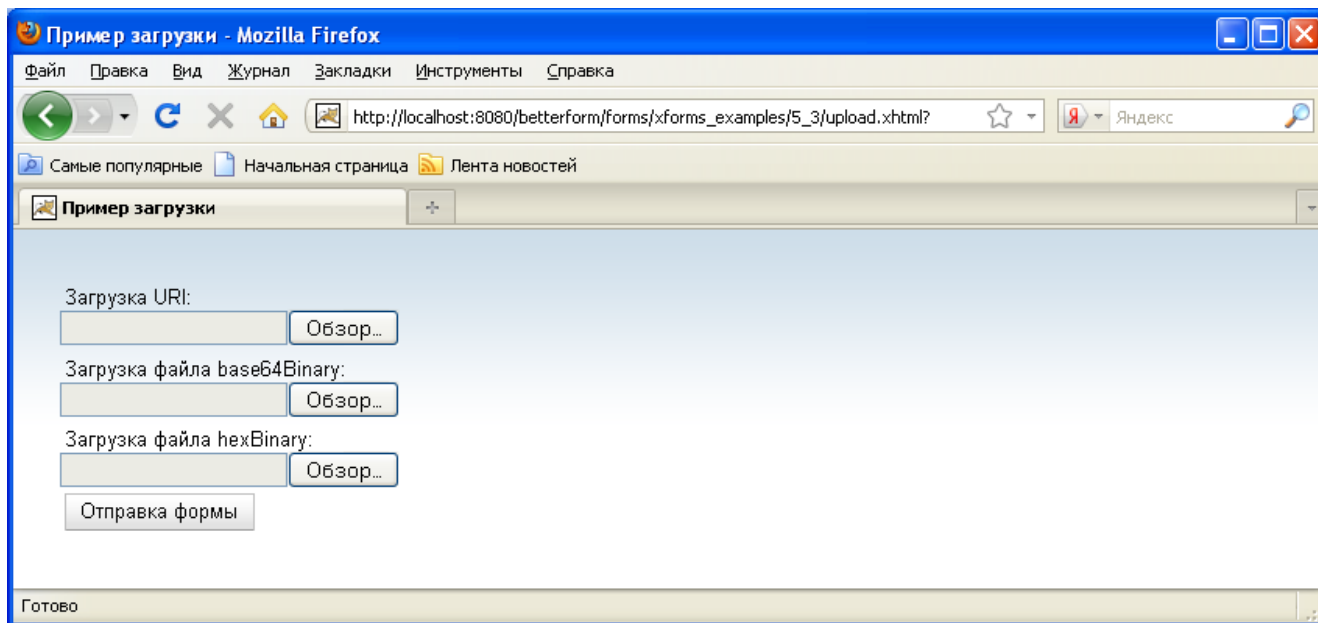


Рис.5.13. Результат выполнения примера 5.3. Файл «upload.xhtml».

### 5.7.6 Элемент range

Большинством XForms-процессоров отображается как шкала, на которой можно выбрать одно значение.

Элемент range содержит атрибуты:

- start=«начальное значение диапазона»
- end=«конечное значение диапазона»
- step=«шаг»

По спецификации может использоваться для данных следующих типов: xsd:duration, xsd:date, xsd:time, xsd:dateTime, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth, xsd:float, xsd:double, xsd:decimal.

Наиболее часто используется для целочисленных данных.

**Пример 5.3. Файл «range.xhtml».**

```

<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример элемента range</title>

    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model>
      <xforms:instance>
        <example xmlns="">
          <int/>
        </example>
      </xforms:instance>

      <xforms:bind nodeset="int" type="xforms:integer"/>
    </xforms:model>
  </head>

  <body>
    <xforms:group>

      <xforms:input ref="int">
        <xforms:label>Введите число:</xforms:label>
      </xforms:input>

      <xforms:range ref="int" start="0" end="10" step="1">
        <xforms:label>Выберите значение:</xforms:label>
      </xforms:range>

    </xforms:group>
  </body>
</html>

```

Результат выполнения примера:

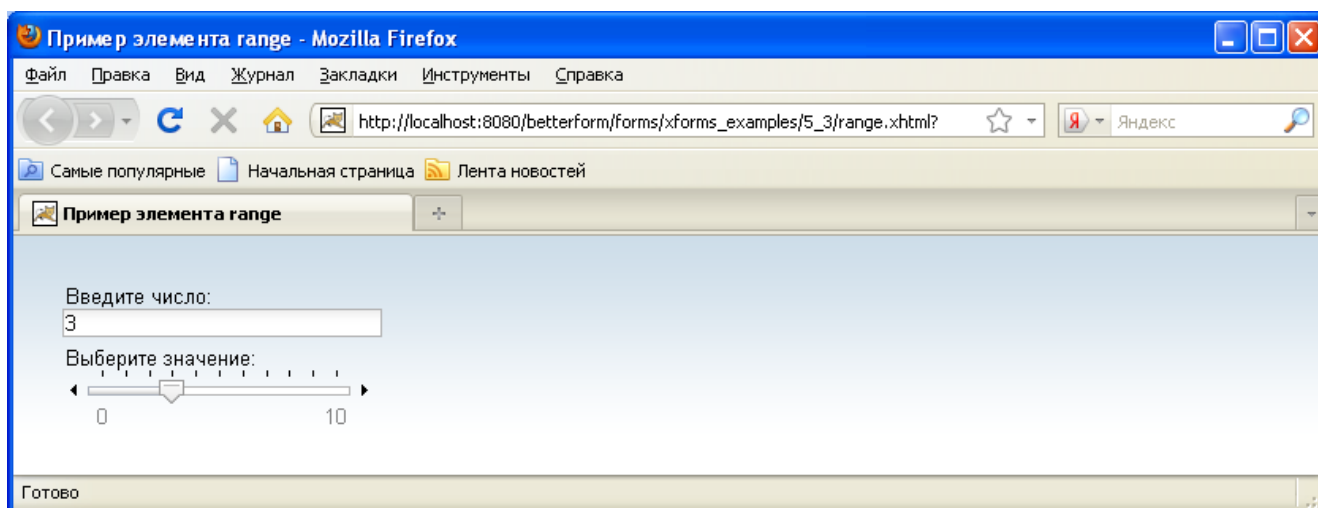


Рис. 5.14. Результат выполнения примера 5.3. Файл «range.xhtml».

### 5.7.7 Элементы задания подсказок и вывода сообщений

Для задания подсказок и вывода сообщений в XForms используются следующие элементы:

- `label` – подпись к элементу, как правило, отображается слева от элемента.
- `help` – справочная информация об элементе.
- `hint` – короткая справка об элементе, может отображаться как всплывающая подсказка.
- `alert` – сообщение об ошибке, может отображаться в виде всплывающей подсказки при ошибке ввода данных в элемент управления.

Разные XForms-процессоры могут отображать эти элементы различными способами.

#### Пример 5.3. Файл «input\_hints.xhtml».

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример элемента input - подсказки</title>
```

```

<!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
<xforms:model>
  <xforms:instance>
    <example xmlns="">
      <text/>
      <date/>
    </example>
  </xforms:instance>

  <xforms:bind nodeset="date" type="xforms:date"/>
  <xforms:submission action="" method="post" id="submit_id"
includenamespacesprefixes=""/>
</xforms:model>
</head>

<body>
  <xforms:group>

    <xforms:input ref="text">
      <xforms:label>(label) Текст подсказки:</xforms:label>
      <xforms:help>(help) Помощь по элементу</xforms:help>
      <xforms:hint>(hint) Всплывающая подсказка</xforms:hint>
      <xforms:alert>(alert) Окно сообщения с
подсказкой</xforms:alert>
    </xforms:input>

    <xforms:input ref="date">
      <xforms:label>(label) Введите дату:</xforms:label>
      <xforms:help>(help) Помощь для элемента ввода
даты</xforms:help>
      <xforms:hint>(hint) Всплывающая подсказка для элемента ввода
даты</xforms:hint>

```

```

<xforms:alert>(alert) Необходимо ввести правильную
дату</xforms:alert>
</xforms:input>

<xforms:submit submission="submit_id">
  <xforms:label>Отправка формы</xforms:label>
</xforms:submit>

</xforms:group>
</body>
</html>

```

Результат выполнения примера:

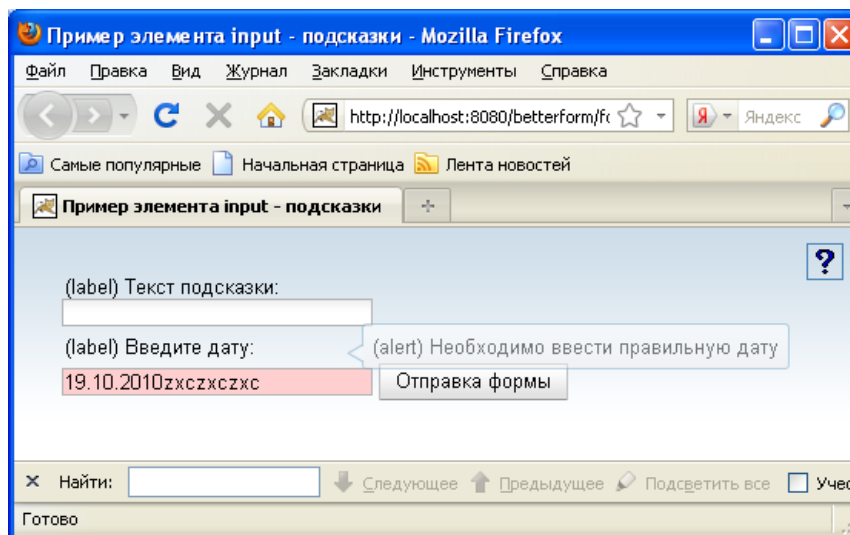


Рис. 5.15. Результат выполнения примера 5.3. Файл «input\_hints.xhtml».

Сообщения можно хранить в модели данных, что бывает полезно при разработке приложений. В этом случае может использоваться способ обращения к данным через атрибут «ref».

### Пример 5.3. Файл «input\_hints\_model.xhtml».

```

<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
<head>
  <title>Пример элемента input - модель</title>

```

`<!-- Пустой префикс пространства имен обязателен для корневого элемента xforms:instance -->`

```
<xforms:model id="model_data">
```

```
  <xforms:instance>
```

```
    <example xmlns="">
```

```
      <text/>
```

```
      <date/>
```

```
    </example>
```

```
  </xforms:instance>
```

```
  <xforms:bind nodeset="date" type="xforms:date"/>
```

```
  <xforms:submission action="" method="post" id="submit_id"
includenamespacesprefixes=""/>
```

```
</xforms:model>
```

`<!-- Сообщения заданы в виде отдельной модели данных, они могут быть загружены из базы данных -->`

```
<xforms:model id="model_message">
```

```
  <xforms:instance>
```

```
    <example xmlns="">
```

```
      <message_label>(label) Введите дату:</message_label>
```

```
      <message_help>(help) Помощь для элемента ввода
даты</message_help>
```

```
      <message_hint>(hint) Всплывающая подсказка для элемента
ввода даты</message_hint>
```

```
      <message_alert>(alert) Необходимо ввести правильную
дату</message_alert>
```

```
    </example>
```

```
  </xforms:instance>
```

```
</xforms:model>
```

```
</head>
```

```
<body>
```

```
  <xforms:group>
```

```

<xforms:input ref="text" model="model_data">
  <xforms:label>(label) Текст подсказки:</xforms:label>
  <xforms:help>(help) Помощь по элементу</xforms:help>
  <xforms:hint>(hint) Всплывающая подсказка</xforms:hint>
  <xforms:alert>(alert) Окно сообщения с
подсказкой</xforms:alert>
</xforms:input>

<xforms:input ref="date" model="model_data">
  <!-- Сообщения заданы в виде ссылок на модель -->
  <xforms:label model="model_message" ref="message_label" />
  <xforms:help model="model_message" ref="message_help"/>
  <xforms:hint model="model_message" ref="message_hint"/>
  <xforms:alert model="model_message" ref="message_alert"/>
</xforms:input>

<!-- Кнопка отправки формы. Атрибут submission ссылается на
элемент xforms:submission в модели -->
<xforms:submit submission="submit_id">
  <xforms:label>Отправка формы</xforms:label>
</xforms:submit>
</xforms:group>
</body>
</html>

```

Результат выполнения совпадает с предыдущим примером.

### 5.7.8 Элемент trigger

Большинством XForms-процессоров отображается как кнопка или гиперссылка.

Обычно на этот элемент оказывает влияние атрибут «appearance». betterFORM отображает элемент «trigger» как кнопку в случае «appearance="full" | "compact"» или как гиперссылку в случае «appearance="minimal"».



### Пример 5.3. Файл «trigger.xhtml».

```

<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример элемента trigger</title>

    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model>
      <xforms:instance>
        <example xmlns="">
          <int/>
        </example>
      </xforms:instance>

      <xforms:bind nodeset="int" type="xforms:integer"/>
    </xforms:model>
  </head>

  <body>
    <xforms:group>

      <xforms:trigger appearance="full">
        <xforms:label>(appearance="full")</xforms:label>
        <xforms:help>(help) Помощь по элементу</xforms:help>
        <xforms:hint>(hint) Всплывающая подсказка</xforms:hint>
        <xforms:alert>(alert) Окно сообщения с
подсказкой</xforms:alert>
      </xforms:trigger>

      <xforms:trigger appearance="compact">
        <xforms:label>(appearance="compact")</xforms:label>
        <xforms:help>(help) Помощь по элементу</xforms:help>

```

```

    <xforms:hint>(hint) Всплывающая подсказка</xforms:hint>
    <xforms:alert>(alert) Окно сообщения с
подсказкой</xforms:alert>
  </xforms:trigger>

  <xforms:trigger appearance="minimal">
    <xforms:label>(appearance="minimal")</xforms:label>
    <xforms:help>(help) Помощь по элементу</xforms:help>
    <xforms:hint>(hint) Всплывающая подсказка</xforms:hint>
    <xforms:alert>(alert) Окно сообщения с
подсказкой</xforms:alert>
  </xforms:trigger>

</xforms:group>
</body>
</html>

```

Результат выполнения примера:

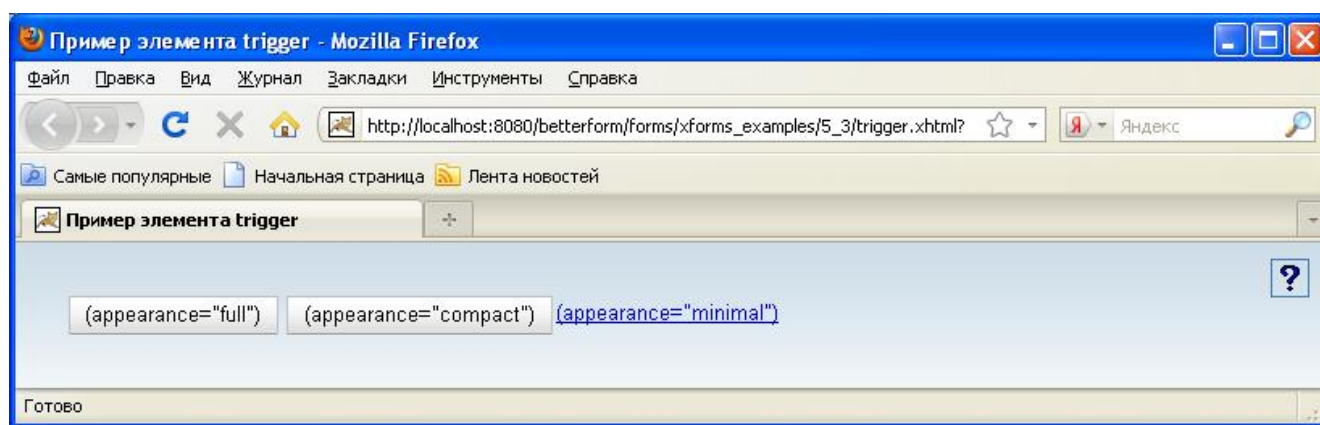


Рис. 5.16. Результат выполнения примера 5.3. Файл «trigger.xhtml».

### 5.7.9 Элемент submit

Большинством XForms-процессоров отображается так же, как элемент trigger (в виде кнопки или гиперссылки в зависимости от атрибута «appearance»).

Но если элемент `trigger` является кнопкой общего вида и реагирует на события, то элемент `submit` используется для завершения редактирования и отправки заполненных данных формы на сервер.

У элемента `submit` должен быть задан атрибут «`submission=id` элемента `submission`». Элемент `submission` в модели данных определяет параметры отправки заполненной формы.

### 5.7.10 Элемент `select`

Элемент `select` позволяет выбирать несколько значений из списка элементов. Элементы списка могут быть заданы в виде констант или в виде данных в модели данных.

Несмотря на то, что список позволяет выбирать множество элементов, привязка результата выбора осуществляется к единственному XML-элементу. Это связано с тем, что выбранные элементы сохраняются в виде XML-списка (тип данных «`xsd:list`»).

Например, если в `select` выбраны элементы с кодами «`val1`» и «`val3`», то XML-элемент данных «`result`» будет выглядеть как «`<result>val1 val3</result>`». Коды выбранных элементов списка будут помещены в XML-тэг `result` и будут разделены пробелами.

На элемент «`select`» оказывает влияние атрибут «`appearance`».

Если у элемента «`select`» указан атрибут «`selection="open"`», то список должен отображаться с полем ввода, в которое можно вводить дополнительные значения.

Для создания элементов списка можно использовать два способа.

1. Элементы списка задаются в виде констант.

В этом случае в элемент «`xforms:select`» помещается элемент «`xforms:choices`», а в него множество элементов «`xforms:item`», которые соответствуют элементам списка.

В элемент «`xforms:item`» помещается элемент «`xforms:value`», содержащий данные (код элемента, который записывается в результат), и элемент «`xforms:label`», содержащий отображаемое значение.

Пример:

```
<xforms:select model="result_model" ref="result">
  <xforms:choices>
    <xforms:item>
      <xforms:label>Значение 1</xforms:label>
      <xforms:value>val1</xforms:value>
    </xforms:item>
    . . .
  </xforms:choices>
</xforms:select>
```

## 2. Элементы списка читаются из модели данных.

В этом случае в элемент «xforms:select» помещается элемент «xforms:itemset», который задает множество элементов списка.

Элемент «xforms:itemset» содержит атрибут «model» для ссылки на модель данных и атрибут «nodeset», в котором указано XPath-выражение, выбирающее множество XML-элементов данных, соответствующих элементам списка.

В элемент «xforms:itemset», как и в предыдущем случае, помещаются элементы «xforms:value» и «xforms:label». Но здесь у них заданы атрибуты «ref», которые выбирают код и отображаемое значение из элементов, выбранных с помощью атрибута «nodeset» элемента «xforms:itemset».

Элементы списка могут читаться из одной модели данных, а результат может сохраняться в другую модель данных.

Пример:

Модели данных:

```
<xforms:model id="result_model">
  <xforms:instance>
    <example xmlns="">
      <result>val1 val3</result>
    </example>
  </xforms:instance>
</xforms:model>
```

```

<xforms:model id="list_data">
  <xforms:instance>
    <example xmlns="">
      <item>
        <text>Значение из модели 1</text>
        <data>val1</data>
      </item>
      . . .
    </example>
  </xforms:instance>
</xforms:model>

```

### Список:

```

<xforms:select model="result_model" ref="result">
  <xforms:itemset model="list_data" nodeset="item">
    <xforms:label model="list_data" ref="text"/>
    <xforms:value model="list_data" ref="data"/>
  </xforms:itemset>
</xforms:select>

```

### Пример 5.3. Файл «select.xhtml».

```

<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример элемента select</title>

```

<!-- Пустой префикс пространства имен обязателен для корневого элемента xforms:instance -->

```

  <xforms:model id="result_model">
    <xforms:instance>
      <example xmlns="">
        <result>val1 val3</result>
      </example>

```

```

    </xforms:instance>
</xforms:model>

<xforms:model id="list_data">
  <xforms:instance>
    <example xmlns="">
      <item>
        <text>Значение из модели 1</text>
        <data>val1</data>
      </item>
      <item>
        <text>Значение из модели 2</text>
        <data>val2</data>
      </item>
      <item>
        <text>Значение из модели 3</text>
        <data>val3</data>
      </item>
    </example>
  </xforms:instance>
</xforms:model>

```

```
</head>
```

```
<body>
```

```
  <xforms:group>
```

```

    <h1>Список констант без возможности добавления
(selection="closed")</h1>

```

```

  <xforms:select model="result_model" ref="result"
appearance="full">
    <xforms:label>appearance="full":</xforms:label>
    <xforms:choices>
      <xforms:item>

```

```

    <xforms:label>Значение 1</xforms:label>
    <xforms:value>val1</xforms:value>
  </xforms:item>
  <xforms:item>
    <xforms:label>Значение 2</xforms:label>
    <xforms:value>val2</xforms:value>
  </xforms:item>
  <xforms:item>
    <xforms:label>Значение 3</xforms:label>
    <xforms:value>val3</xforms:value>
  </xforms:item>
</xforms:choices>
</xforms:select>

```

```

<xforms:select model="result_model" ref="result"
appearance="compact">
  <xforms:label>appearance="compact":</xforms:label>
  <xforms:choices>
    <xforms:item>
      <xforms:label>Значение 1</xforms:label>
      <xforms:value>val1</xforms:value>
    </xforms:item>
    <xforms:item>
      <xforms:label>Значение 2</xforms:label>
      <xforms:value>val2</xforms:value>
    </xforms:item>
    <xforms:item>
      <xforms:label>Значение 3</xforms:label>
      <xforms:value>val3</xforms:value>
    </xforms:item>
  </xforms:choices>
</xforms:select>

```

```

<xforms:select model="result_model" ref="result"
appearance="minimal">

```

```

<xforms:label>appearance="minimal":</xforms:label>
<xforms:choices>
  <xforms:item>
    <xforms:label>Значение 1</xforms:label>
    <xforms:value>val1</xforms:value>
  </xforms:item>
  <xforms:item>
    <xforms:label>Значение 2</xforms:label>
    <xforms:value>val2</xforms:value>
  </xforms:item>
  <xforms:item>
    <xforms:label>Значение 3</xforms:label>
    <xforms:value>val3</xforms:value>
  </xforms:item>
</xforms:choices>
</xforms:select>

```

```

<h1>Список констант с возможностью добавления
(selection="open")</h1>

```

```

<xforms:select model="result_model" ref="result"
selection="open" appearance="full">
  <xforms:label>appearance="full":</xforms:label>
  <xforms:choices>
    <xforms:item>
      <xforms:label>Значение 1</xforms:label>
      <xforms:value>val1</xforms:value>
    </xforms:item>
    <xforms:item>
      <xforms:label>Значение 2</xforms:label>
      <xforms:value>val2</xforms:value>
    </xforms:item>
    <xforms:item>
      <xforms:label>Значение 3</xforms:label>
      <xforms:value>val3</xforms:value>
    </xforms:item>
  </xforms:choices>
</xforms:select>

```



```

    </xforms:item>
  </xforms:choices>
</xforms:select>

```

```

<xforms:select model="result_model" ref="result"
selection="open" appearance="compact">
  <xforms:label>appearance="compact":</xforms:label>
  <xforms:choices>
    <xforms:item>
      <xforms:label>Значение 1</xforms:label>
      <xforms:value>val1</xforms:value>
    </xforms:item>
    <xforms:item>
      <xforms:label>Значение 2</xforms:label>
      <xforms:value>val2</xforms:value>
    </xforms:item>
    <xforms:item>
      <xforms:label>Значение 3</xforms:label>
      <xforms:value>val3</xforms:value>
    </xforms:item>
  </xforms:choices>
</xforms:select>

```

```

<xforms:select model="result_model" ref="result"
selection="open" appearance="minimal">
  <xforms:label>appearance="minimal":</xforms:label>
  <xforms:choices>
    <xforms:item>
      <xforms:label>Значение 1</xforms:label>
      <xforms:value>val1</xforms:value>
    </xforms:item>
    <xforms:item>
      <xforms:label>Значение 2</xforms:label>
      <xforms:value>val2</xforms:value>
    </xforms:item>
  </xforms:choices>

```

```

<xforms:item>
  <xforms:label>Значение 3</xforms:label>
  <xforms:value>val3</xforms:value>
</xforms:item>
</xforms:choices>
</xforms:select>

```

```

<h1>Список с привязкой данных без возможности добавления
(selection="closed")</h1>

```

```

<xforms:select model="result_model" ref="result"
appearance="full">
  <xforms:label>appearance="full":</xforms:label>
  <xforms:itemset model="list_data" nodeset="item">
    <xforms:label model="list_data" ref="text"/>
    <!-- Вместо элемента value здесь может быть использован
элемент copy -->
    <xforms:value model="list_data" ref="data"/>
  </xforms:itemset>
</xforms:select>

```

```

<xforms:select model="result_model" ref="result"
appearance="compact">
  <xforms:label>appearance="compact":</xforms:label>
  <xforms:itemset model="list_data" nodeset="item">
    <xforms:label model="list_data" ref="text"/>
    <!-- Вместо элемента value здесь может быть использован
элемент copy -->
    <xforms:value model="list_data" ref="data"/>
  </xforms:itemset>
</xforms:select>

```

```

<xforms:select model="result_model" ref="result"
appearance="minimal">
  <xforms:label>appearance="minimal":</xforms:label>

```

```

<xforms:itemset model="list_data" nodeset="item">
  <xforms:label model="list_data" ref="text"/>
  <!-- Вместо элемента value здесь может быть использован
элемент copy -->
  <xforms:value model="list_data" ref="data"/>
</xforms:itemset>
</xforms:select>

```

```

<h1>Список с привязкой данных с возможностью добавления
(selection="open")</h1>

```

```

<xforms:select model="result_model" ref="result"
selection="open" appearance="full">
  <xforms:label>appearance="full":</xforms:label>
  <xforms:itemset model="list_data" nodeset="item">
    <xforms:label model="list_data" ref="text"/>
    <!-- Вместо элемента value здесь может быть использован
элемент copy -->
    <xforms:value model="list_data" ref="data"/>
  </xforms:itemset>
</xforms:select>

```

```

<xforms:select model="result_model" ref="result"
selection="open" appearance="compact">
  <xforms:label>appearance="compact":</xforms:label>
  <xforms:itemset model="list_data" nodeset="item">
    <xforms:label model="list_data" ref="text"/>
    <!-- Вместо элемента value здесь может быть использован
элемент copy -->
    <xforms:value model="list_data" ref="data"/>
  </xforms:itemset>
</xforms:select>

```

```

<xforms:select model="result_model" ref="result"
selection="open" appearance="minimal">

```

```

<xforms:label>appearance="minimal":</xforms:label>
<xforms:itemset model="list_data" nodeset="item">
  <xforms:label model="list_data" ref="text"/>
  <!-- Вместо элемента value здесь может быть использован
элемент copy -->
  <xforms:value model="list_data" ref="data"/>
</xforms:itemset>
</xforms:select>

</xforms:group>
</body>
</html>

```

Результат выполнения примера:

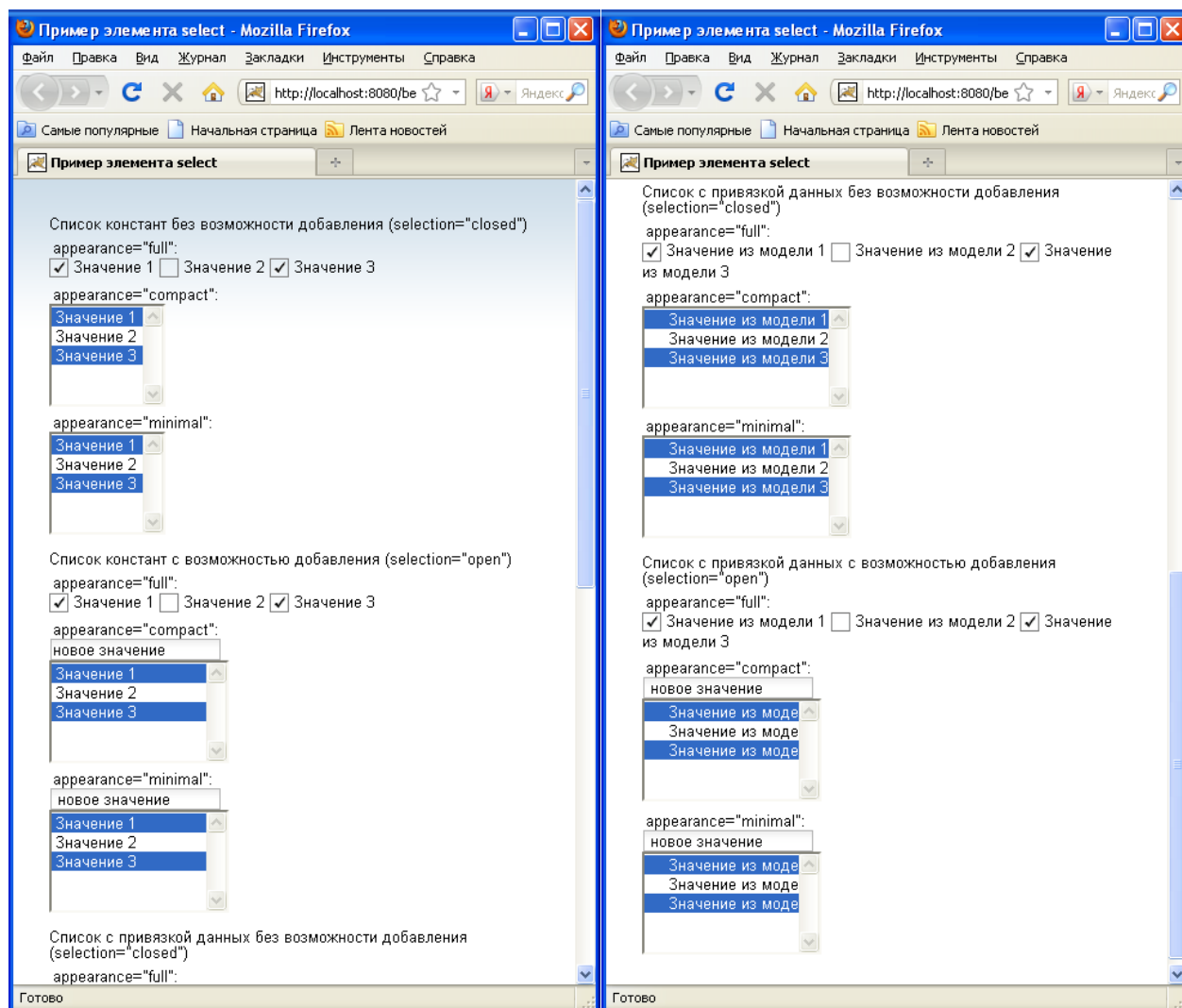


Рис. 5.17. Результат выполнения примера 5.3. Файл «select.xhtml».

В случае использования элементов «xforms:itemset» и «xforms:value» существует проблема, связанная с тем, что элемент «xforms:value» позволяет выбирать только простые значения данных, но не позволяет выбрать поддерево.

Для выбора поддерева вместо элемента «xforms:value» используется элемент «xforms:copy».

В следующем примере в результат помещается XML-элемент с вложенными данными, который копируется с помощью «xforms:copy».

### **Пример 5.3. Файл «select\_copy.xhtml».**

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример элемента copy</title>

    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model id="result_model">
      <xforms:instance>
        <example xmlns="">
          <result/>
        </example>
      </xforms:instance>
    </xforms:model>

    <xforms:model id="list_data">
      <xforms:instance>
        <example xmlns="">
          <item>
            <text>Значение из модели 1</text>
            <data>val1</data>
          </item>
          <item>
            <text>Значение из модели 2</text>
```

[Оглавление](#)

```

        <data>val2</data>
    </item>
    <item>
        <text>Значение из модели 3</text>
        <data>val3</data>
    </item>
</example>
</xforms:instance>
</xforms:model>
</head>

<body>
    <xforms:group>

        <xforms:select model="result_model" ref="result">
            <xforms:label>Выберите элемент:</xforms:label>
            <xforms:itemset model="list_data" nodeset="item">
                <xforms:label model="list_data" ref="text"/>
                <!-- Элемент item (вместе с вложенными данными) будет
помещен в результат. XPath-выражение "." является контекстным по
отношению к nodeset="item", то есть "." выберет элемент "item". -->
                <xforms:copy model="list_data" ref="."/>
            </xforms:itemset>
        </xforms:select>

        <!-- Выбранный элемент списка, который был помещен в
результатирующую модель -->
        <xforms:output model="result_model" value="result/item/text">
            <xforms:label>Выбранный элемент:</xforms:label>
        </xforms:output>

    </xforms:group>
</body>
</html>

```

Результат выполнения примера:

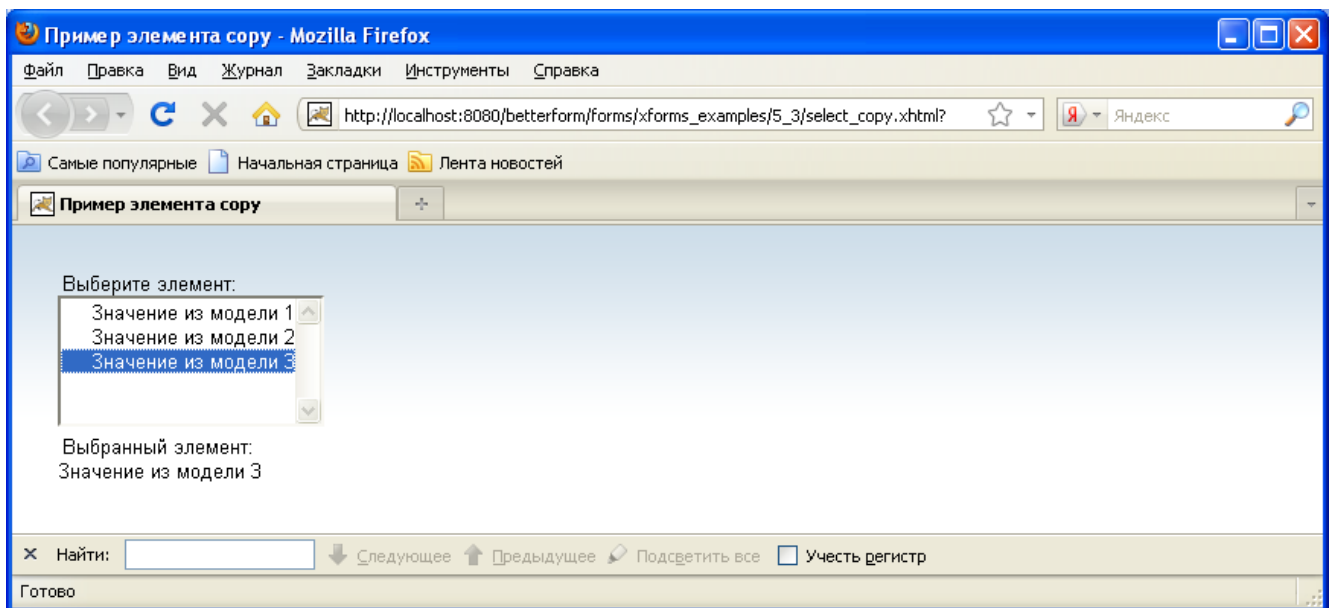


Рис. 5.18. Результат выполнения примера 5.3. Файл «select\_copy.xhtml».

### 5.7.11 Элемент select1

Отличие этого элемента от select в том, что select1 позволяет выбирать единственное значение из списка элементов.

#### Пример 5.3. Файл «select1.xhtml».

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример элемента select1</title>

    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model id="result_model">
      <xforms:instance>
        <example xmlns="">
          <result>val2</result>
        </example>
      </xforms:instance>
    </xforms:model>
```

[Оглавление](#)

```

<xforms:model id="list_data">
  <xforms:instance>
    <example xmlns="">
      <item>
        <text>Значение из модели 1</text>
        <data>val1</data>
      </item>
      <item>
        <text>Значение из модели 2</text>
        <data>val2</data>
      </item>
      <item>
        <text>Значение из модели 3</text>
        <data>val3</data>
      </item>
    </example>
  </xforms:instance>
</xforms:model>
</head>

<body>
  <xforms:group>
    <h1>Список констант без возможности добавления
(selection="closed")</h1>

    <xforms:select1 model="result_model" ref="result"
appearance="full">
      <xforms:label>appearance="full":</xforms:label>
      <xforms:choices>
        <xforms:item>
          <xforms:label>Значение 1</xforms:label>
          <xforms:value>val1</xforms:value>
        </xforms:item>
        <xforms:item>

```



```

    <xforms:label>Значение 2</xforms:label>
    <xforms:value>val2</xforms:value>
  </xforms:item>
  <xforms:item>
    <xforms:label>Значение 3</xforms:label>
    <xforms:value>val3</xforms:value>
  </xforms:item>
</xforms:choices>
</xforms:select1>

```

```

<xforms:select1 model="result_model" ref="result"
appearance="compact">
  <xforms:label>appearance="compact":</xforms:label>
  <xforms:choices>
    <xforms:item>
      <xforms:label>Значение 1</xforms:label>
      <xforms:value>val1</xforms:value>
    </xforms:item>
    <xforms:item>
      <xforms:label>Значение 2</xforms:label>
      <xforms:value>val2</xforms:value>
    </xforms:item>
    <xforms:item>
      <xforms:label>Значение 3</xforms:label>
      <xforms:value>val3</xforms:value>
    </xforms:item>
  </xforms:choices>
</xforms:select1>

```

```

<xforms:select1 model="result_model" ref="result"
appearance="minimal">
  <xforms:label>appearance="minimal":</xforms:label>
  <xforms:choices>
    <xforms:item>
      <xforms:label>Значение 1</xforms:label>

```

```

        <xforms:value>val1</xforms:value>
    </xforms:item>
    <xforms:item>
        <xforms:label>Значение 2</xforms:label>
        <xforms:value>val2</xforms:value>
    </xforms:item>
    <xforms:item>
        <xforms:label>Значение 3</xforms:label>
        <xforms:value>val3</xforms:value>
    </xforms:item>
</xforms:choices>
</xforms:select1>

```

```

<h1>Список констант с возможностью добавления
(selection="open")</h1>

```

```

<xforms:select1 model="result_model" ref="result"
selection="open" appearance="full">
    <xforms:label>appearance="full":</xforms:label>
    <xforms:choices>
        <xforms:item>
            <xforms:label>Значение 1</xforms:label>
            <xforms:value>val1</xforms:value>
        </xforms:item>
        <xforms:item>
            <xforms:label>Значение 2</xforms:label>
            <xforms:value>val2</xforms:value>
        </xforms:item>
        <xforms:item>
            <xforms:label>Значение 3</xforms:label>
            <xforms:value>val3</xforms:value>
        </xforms:item>
    </xforms:choices>
</xforms:select1>

```

```

<xforms:select1 model="result_model" ref="result"
selection="open" appearance="compact">
  <xforms:label>appearance="compact":</xforms:label>
  <xforms:choices>
    <xforms:item>
      <xforms:label>Значение 1</xforms:label>
      <xforms:value>val1</xforms:value>
    </xforms:item>
    <xforms:item>
      <xforms:label>Значение 2</xforms:label>
      <xforms:value>val2</xforms:value>
    </xforms:item>
    <xforms:item>
      <xforms:label>Значение 3</xforms:label>
      <xforms:value>val3</xforms:value>
    </xforms:item>
  </xforms:choices>
</xforms:select1>

```

```

<xforms:select1 model="result_model" ref="result"
selection="open" appearance="minimal">
  <xforms:label>appearance="minimal":</xforms:label>
  <xforms:choices>
    <xforms:item>
      <xforms:label>Значение 1</xforms:label>
      <xforms:value>val1</xforms:value>
    </xforms:item>
    <xforms:item>
      <xforms:label>Значение 2</xforms:label>
      <xforms:value>val2</xforms:value>
    </xforms:item>
    <xforms:item>
      <xforms:label>Значение 3</xforms:label>
      <xforms:value>val3</xforms:value>
    </xforms:item>
  </xforms:choices>
</xforms:select1>

```

```

    </xforms:choices>
</xforms:select1>

```

```

<h1>Список с привязкой данных без возможности добавления
(selection="closed")</h1>

```

```

<xforms:select1 model="result_model" ref="result"
appearance="full">
    <xforms:label>appearance="full":</xforms:label>
    <xforms:itemset model="list_data" nodeset="item">
        <xforms:label model="list_data" ref="text"/>
        <!-- Вместо элемента value здесь может быть использован
элемент copy -->
        <xforms:value model="list_data" ref="data"/>
    </xforms:itemset>
</xforms:select1>

```

```

<xforms:select1 model="result_model" ref="result"
appearance="compact">
    <xforms:label>appearance="compact":</xforms:label>
    <xforms:itemset model="list_data" nodeset="item">
        <xforms:label model="list_data" ref="text"/>
        <!-- Вместо элемента value здесь может быть использован
элемент copy -->
        <xforms:value model="list_data" ref="data"/>
    </xforms:itemset>
</xforms:select1>

```

```

<xforms:select1 model="result_model" ref="result"
appearance="minimal">
    <xforms:label>appearance="minimal":</xforms:label>
    <xforms:itemset model="list_data" nodeset="item">
        <xforms:label model="list_data" ref="text"/>
        <!-- Вместо элемента value здесь может быть использован
элемент copy -->

```

```

    <xforms:value model="list_data" ref="data"/>
  </xforms:itemset>
</xforms:select1>

```

<h1>Список с привязкой данных с возможностью добавления  
(selection="open")</h1>

```

<xforms:select1 model="result_model" ref="result"
selection="open" appearance="full">
  <xforms:label>appearance="full":</xforms:label>
  <xforms:itemset model="list_data" nodeset="item">
    <xforms:label model="list_data" ref="text"/>
    <!-- Вместо элемента value здесь может быть использован
элемент copy -->
    <xforms:value model="list_data" ref="data"/>
  </xforms:itemset>
</xforms:select1>

```

```

<xforms:select1 model="result_model" ref="result"
selection="open" appearance="compact">
  <xforms:label>appearance="compact":</xforms:label>
  <xforms:itemset model="list_data" nodeset="item">
    <xforms:label model="list_data" ref="text"/>
    <!-- Вместо элемента value здесь может быть использован
элемент copy -->
    <xforms:value model="list_data" ref="data"/>
  </xforms:itemset>
</xforms:select1>

```

```

<xforms:select1 model="result_model" ref="result"
selection="open" appearance="minimal">
  <xforms:label>appearance="minimal":</xforms:label>
  <xforms:itemset model="list_data" nodeset="item">
    <xforms:label model="list_data" ref="text"/>
  </xforms:itemset>
</xforms:select1>

```

```
<!-- Вместо элемента value здесь может быть использован
элемент copy -->
```

```
<xforms:value model="list_data" ref="data"/>
```

```
</xforms:itemset>
```

```
</xforms:select1>
```

```
</xforms:group>
```

```
</body>
```

```
</html>
```

Результат выполнения примера:

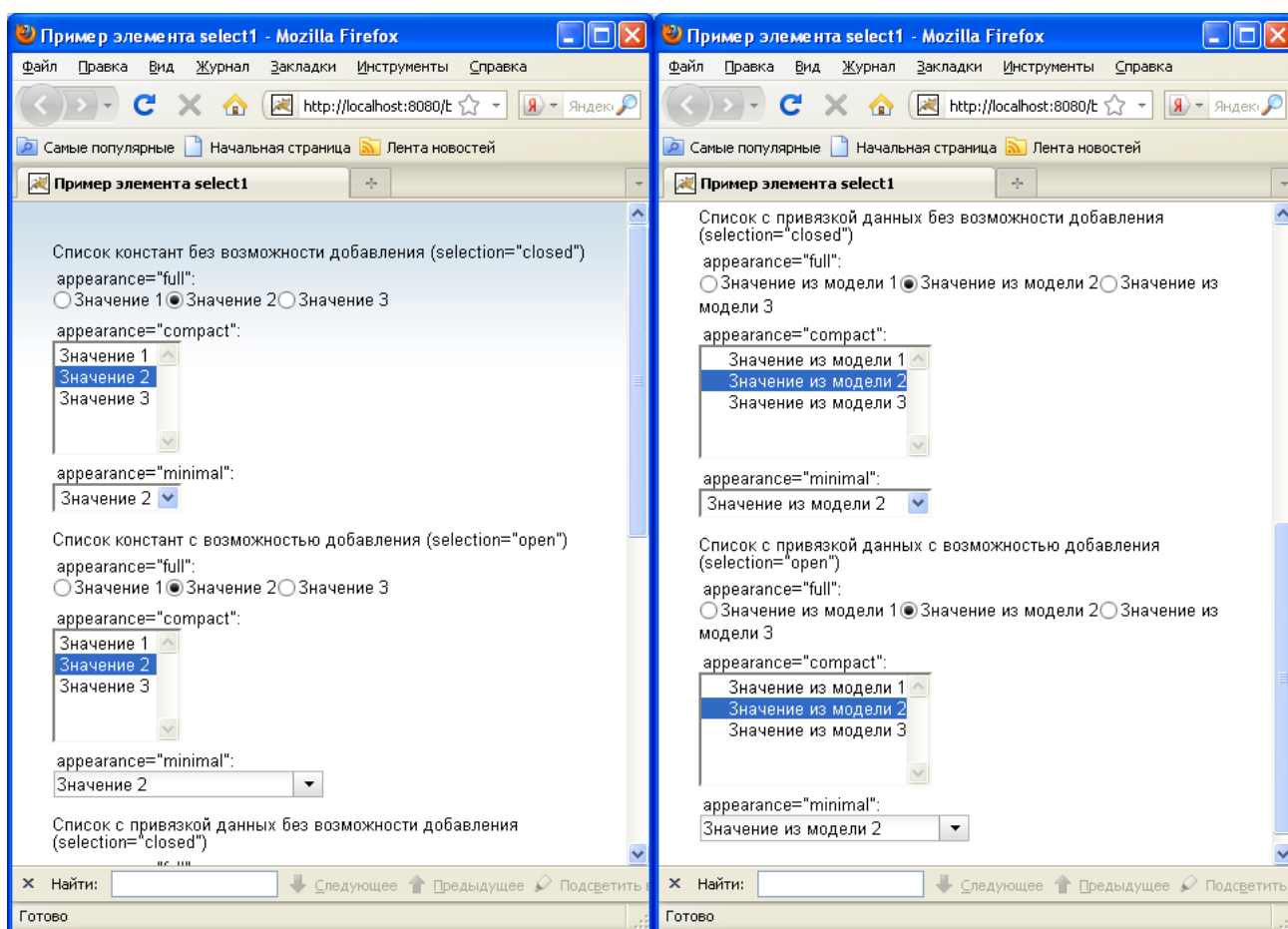


Рис.5.19. Результат выполнения примера 5.3. Файл «select1.xhtml».

### 5.7.12 Элемент group

Используется как простой контейнер для элементов управления. Элементы «group» могут быть вложены друг в друга.

На элемент «group» оказывает влияние атрибут «appearance».

[Оглавление](#)

Элемент «group» может быть привязан к XML-элементам данных с помощью атрибута «ref».

### Пример 5.3. Файл «group.xhtml».

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример элемента group</title>

    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model>
      <xforms:instance>
        <example xmlns="">
          <text1/>
          <text2/>
          <text3/>
        </example>
      </xforms:instance>
    </xforms:model>
  </head>

  <body>
    <xforms:group appearance="full">
      <xforms:label>Заголовок группы
(apppearance="full")</xforms:label>

      <xforms:input ref="text1">
        <xforms:label>Введите текст 1:</xforms:label>
      </xforms:input>

      <xforms:input ref="text2">
        <xforms:label>Введите текст 2:</xforms:label>
      </xforms:input>
```

```

    <xforms:input ref="text3">
      <xforms:label>Введите текст 3:</xforms:label>
    </xforms:input>
  </xforms:group>
  <hr/>
  <xforms:group appearance="compact">
    <xforms:label>Заголовок группы
(appearance="compact")</xforms:label>

```

```

    <xforms:input ref="text1">
      <xforms:label>Введите текст 1:</xforms:label>
    </xforms:input>
    <xforms:input ref="text2">
      <xforms:label>Введите текст 2:</xforms:label>
    </xforms:input>
    <xforms:input ref="text3">
      <xforms:label>Введите текст 3:</xforms:label>
    </xforms:input>
  </xforms:group>
  <hr/>

```

```

  <xforms:group appearance="minimal">
    <xforms:label>Заголовок группы
(appearance="minimal")</xforms:label>

```

```

    <xforms:input ref="text1">
      <xforms:label>Введите текст 1:</xforms:label>
    </xforms:input>
    <xforms:input ref="text2">
      <xforms:label>Введите текст 2:</xforms:label>
    </xforms:input>
    <xforms:input ref="text3">
      <xforms:label>Введите текст 3:</xforms:label>
    </xforms:input>
  </xforms:group>

```



```
</body>
</html>
```

Результат выполнения примера:

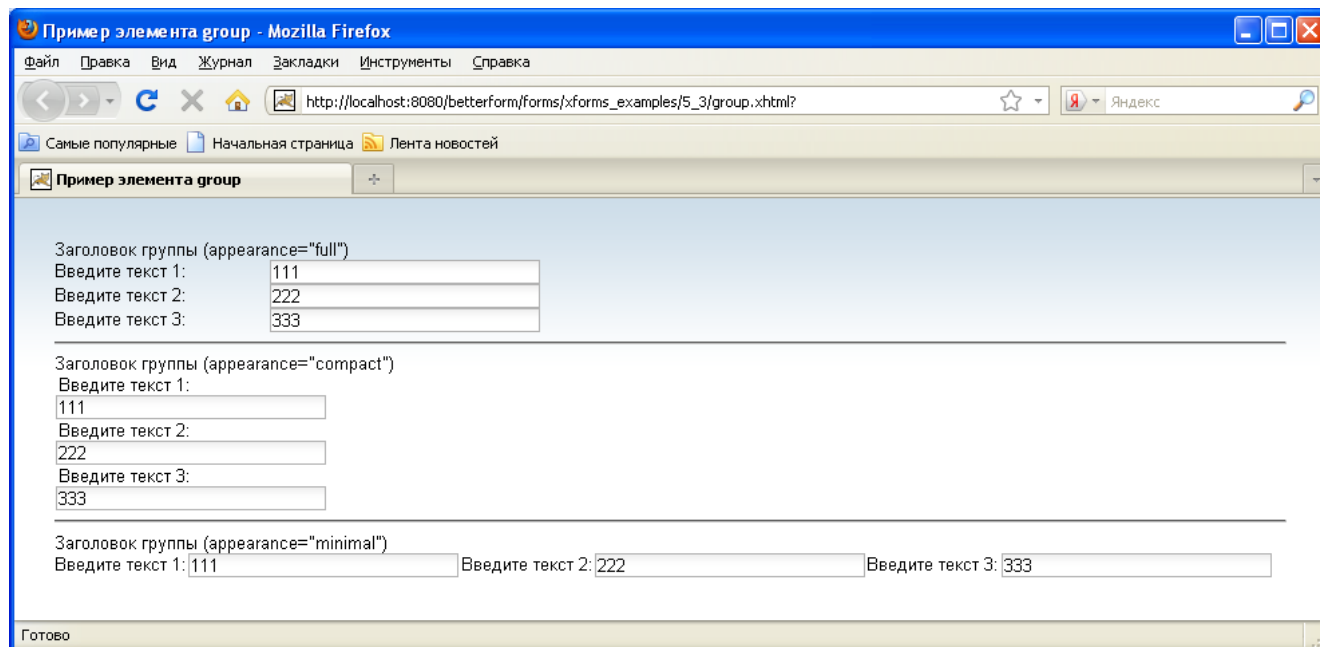


Рис. 5.20. Результат выполнения примера 5.3. Файл «group.xhtml».

### 5.7.13 Элемент switch

Этот элемент позволяет группировать элементы управления во «вкладки» и переключаться между вкладками.

Вкладка задается с помощью элемента «xforms:case», у которого должен быть задан обязательный атрибут «id» для ссылки на вкладку. Элементы управления, относящиеся к вкладке, вложены в элемент «xforms:case».

Активной (отображаемой для пользователя) может быть только одна вкладка, по умолчанию это первая вкладка. Чтобы сделать активной не первую вкладку, у соответствующего элемента «xforms:case» необходимо указать атрибут «selected="true"».

Элемент «switch» позволяет переключать вкладки в результате возникновения события (например, при нажатии на кнопку). В обработчик события помещается элемент «xforms:toggle» с атрибутом «case="id элемента

case"», который переключает вкладку, то есть делает активным указанный элемент «xforms:case».

В элементе «xforms:toggle» не указывается идентификатор элемента «switch», потому что это не нужно, идентификаторы элементов «case» уникальны и по ним можно определить соответствующий элемент «switch».

В элементе «xforms:toggle» вместо атрибута «case» может быть использован вложенный элемент «xforms:case» с атрибутом «value», который позволяет динамически формировать идентификатор вкладки для переключения.

Элемент «switch» не предназначен для того, чтобы переключаться между вкладками на основе изменения значений данных. Для этого может быть использован элемент «group» с атрибутом «ref».

### **Пример 5.3. Файл «switch.xhtml».**

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример элемента switch</title>
```

```
    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
```

```
    <xforms:model id="result_model">
      <xforms:instance>
        <example xmlns="">
          <result>val1</result>
          <num>3</num>
        </example>
      </xforms:instance>
    </xforms:model>
```

```
    <xforms:model id="list_data">
      <xforms:instance>
        <example xmlns="">
```

```

    <item>
      <text>Значение из модели 1</text>
      <data>val1</data>
    </item>
    <item>
      <text>Значение из модели 2</text>
      <data>val2</data>
    </item>
    <item>
      <text>Значение из модели 3</text>
      <data>val3</data>
    </item>
  </example>
</xforms:instance>
</xforms:model>
</head>

<body>
  <xforms:group>

    <!-- Кнопки для переключения -->
    <xforms:trigger>
      <xforms:label>select (appearance="full")</xforms:label>
      <xforms:toggle ev:event="DOMActivate" case="select_1"/>
    </xforms:trigger>

```

Так как у элемента «xforms:toggle» задан атрибут «ev:event="DOMActivate"», то он является обработчиком «стандартного» события для элемента верхнего уровня «xforms:trigger».

```

<xforms:trigger>
  <xforms:label>select (appearance="compact")</xforms:label>
  <xforms:toggle ev:event="DOMActivate" case="select_2"/>
</xforms:trigger>

```

```

<xforms:trigger>
  <xforms:label>select (appearance="minimal")</xforms:label>
  <xforms:toggle ev:event="DOMActivate">
    <!-- id case может быть определено динамически -->
    <xforms:case value="concat('select_', '3')" />
  </xforms:toggle>
</xforms:trigger>

```

```

<!-- Переключатель -->

```

```

<xforms:switch>

```

```

  <xforms:case id="select_1">

```

```

    <xforms:select model="result_model" ref="result"
appearance="full">

```

```

      <xforms:label>appearance="full":</xforms:label>

```

```

      <xforms:itemset model="list_data" nodeset="item">

```

```

        <xforms:label model="list_data" ref="text"/>

```

```

        <xforms:value model="list_data" ref="data"/>

```

```

      </xforms:itemset>

```

```

    </xforms:select>

```

```

  </xforms:case>

```

```

  <xforms:case id="select_2" selected="true">

```

```

    <xforms:select model="result_model" ref="result"
appearance="compact">

```

```

      <xforms:label>appearance="compact":</xforms:label>

```

```

      <xforms:itemset model="list_data" nodeset="item">

```

```

        <xforms:label model="list_data" ref="text"/>

```

```

        <xforms:value model="list_data" ref="data"/>

```

```

      </xforms:itemset>

```

```

    </xforms:select>

```

```

  </xforms:case>

```

```

  <xforms:case id="select_3">

```

```

<xforms:select model="result_model" ref="result"
appearance="minimal">
  <xforms:label>appearance="minimal":</xforms:label>
  <xforms:itemset model="list_data" nodeset="item">
    <xforms:label model="list_data" ref="text"/>
    <xforms:value model="list_data" ref="data"/>
  </xforms:itemset>
</xforms:select>
</xforms:case>

</xforms:switch>
</xforms:group>
</body>
</html>

```

Результат выполнения примера:

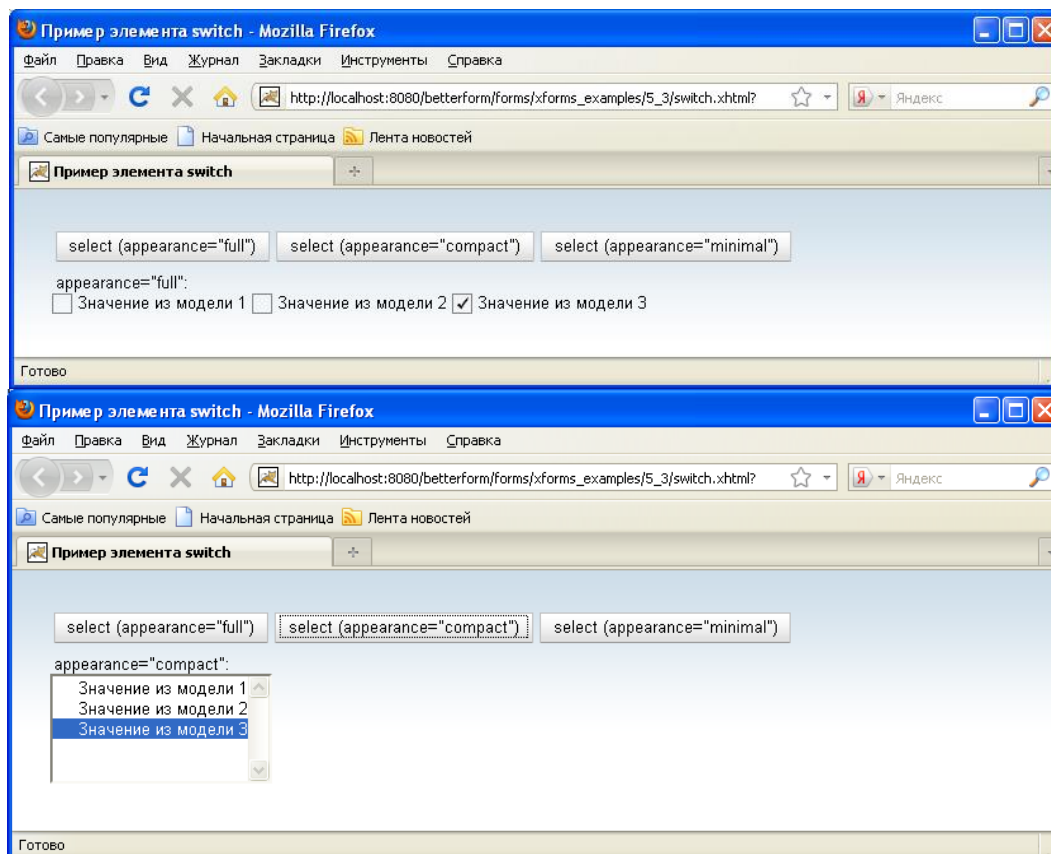


Рис. 5.21. Результат выполнения примера 5.3. Файл «switch.xhtml».

В следующем примере с помощью элемента «switch» реализован прототип «мастера», который позволяет последовательно вводить данные в форму с использованием вкладок и кнопок «далее» и «возврат».

**Пример 5.3. Файл «switch\_prev\_next.xhtml».**

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример мастера на основе switch</title>

    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model id="result_model">
      <xforms:instance>
        <example xmlns="">
          <result>val1</result>
          <num>3</num>
        </example>
      </xforms:instance>
    </xforms:model>

    <xforms:model id="list_data">
      <xforms:instance>
        <example xmlns="">
          <item>
            <text>Значение из модели 1</text>
            <data>val1</data>
          </item>
          <item>
            <text>Значение из модели 2</text>
            <data>val2</data>
          </item>
          <item>
```

```

        <text>Значение из модели 3</text>
        <data>val3</data>
    </item>
</example>
</xforms:instance>
</xforms:model>
</head>

<body>
    <xforms:group>

        <xforms:switch>
            <xforms:case id="select_1" selected="true">
                <xforms:input model="list_data" ref="//item[1]/text">
                    <xforms:label>Первый параметр:</xforms:label>
                </xforms:input>

                <xforms:trigger>
                    <xforms:label>Далее</xforms:label>
                    <xforms:toggle ev:event="DOMActivate" case="select_2"/>
                </xforms:trigger>
            </xforms:case>

            <xforms:case id="select_2">
                <xforms:input model="list_data" ref="//item[2]/text">
                    <xforms:label>Второй параметр:</xforms:label>
                </xforms:input>

                <xforms:trigger>
                    <xforms:label>Возврат</xforms:label>
                    <xforms:toggle ev:event="DOMActivate" case="select_1"/>
                </xforms:trigger>

                <xforms:trigger>
                    <xforms:label>Далее</xforms:label>

```

```

        <xforms:toggle ev:event="DOMActivate" case="select_3"/>
    </xforms:trigger>
</xforms:case>

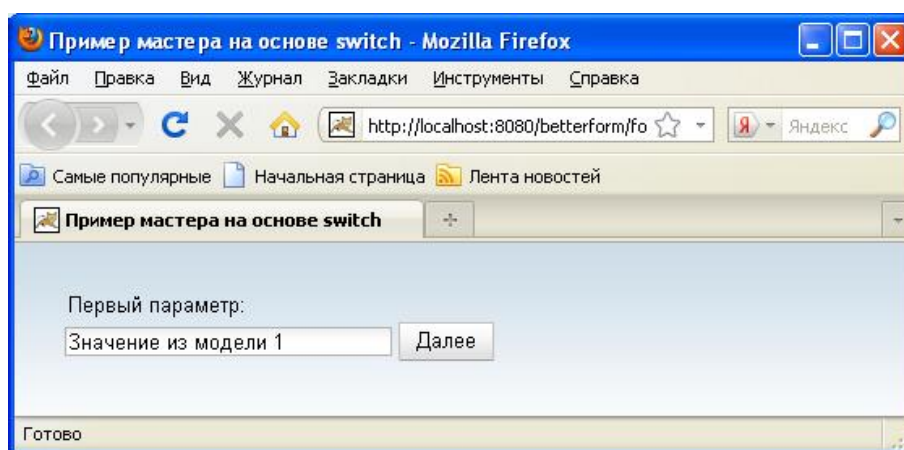
<xforms:case id="select_3">
    <xforms:input model="list_data" ref="//item[3]/text">
        <xforms:label>Третий параметр:</xforms:label>
    </xforms:input>

    <xforms:trigger>
        <xforms:label>Возврат</xforms:label>
        <xforms:toggle ev:event="DOMActivate" case="select_2"/>
    </xforms:trigger>

    <xforms:trigger>
        <xforms:label>Подтверждение</xforms:label>
    </xforms:trigger>
</xforms:case>
</xforms:switch>
</xforms:group>
</body>
</html>

```

Результат выполнения примера:





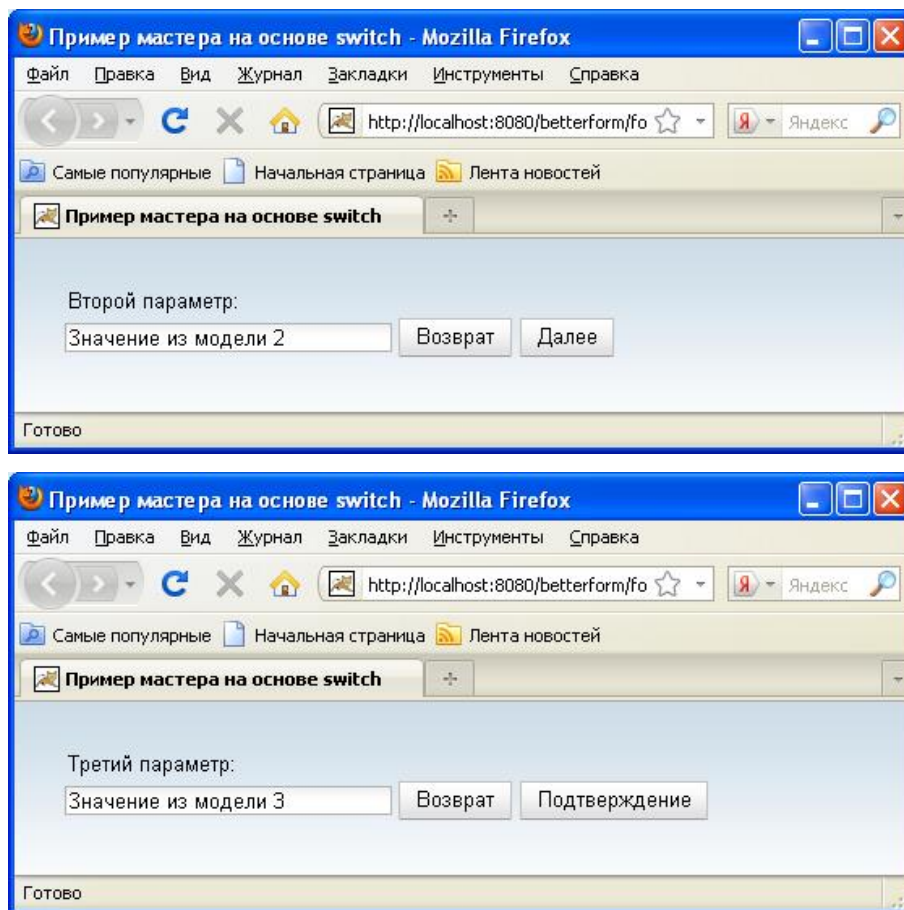


Рис. 5.22. Результат выполнения примера 5.3. Файл «switch\_prev\_next.xhtml».

#### 5.7.14 Элемент repeat

Позволяет обрабатывать повторяющиеся данные, является аналогом цикла.

Основным атрибутом является атрибут «nodeset=XPath-выражение», который позволяет указать повторяющиеся элементы данных.

Атрибуты «startindex = номер первого отображаемого элемента (по умолчанию 1)» и «number = количество отображаемых элементов» позволяют отображать фрагмент множества данных. Поддерживаются не всеми XForms-процессорами.

#### Пример 5.3. Файл «repeat\_simple\_1.xhtml».

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
```

```
<head>
  <title>Пример элемента repeat</title>
```

<!-- Пустой префикс пространства имен обязателен для корневого элемента xforms:instance -->

```
  <xforms:model>
    <xforms:instance>
      <example xmlns="">
        <item>1</item>
        <item>2</item>
        <item>3</item>
        <item>4</item>
        <item>5</item>
      </example>
    </xforms:instance>

  </xforms:model>
```

```
</head>
```

```
<body>
```

```
  <xforms:repeat nodeset="item">
```

Осуществляется перебор всех XML-элементов «item», поле ввода «xforms:input» отображается для каждого XML-элемента «item».

```
    <xforms:input ref=".">
      <xforms:label>Введите текст:</xforms:label>
    </xforms:input>
  </xforms:repeat>
```

```
</body>
```

```
</html>
```

Результат выполнения примера:

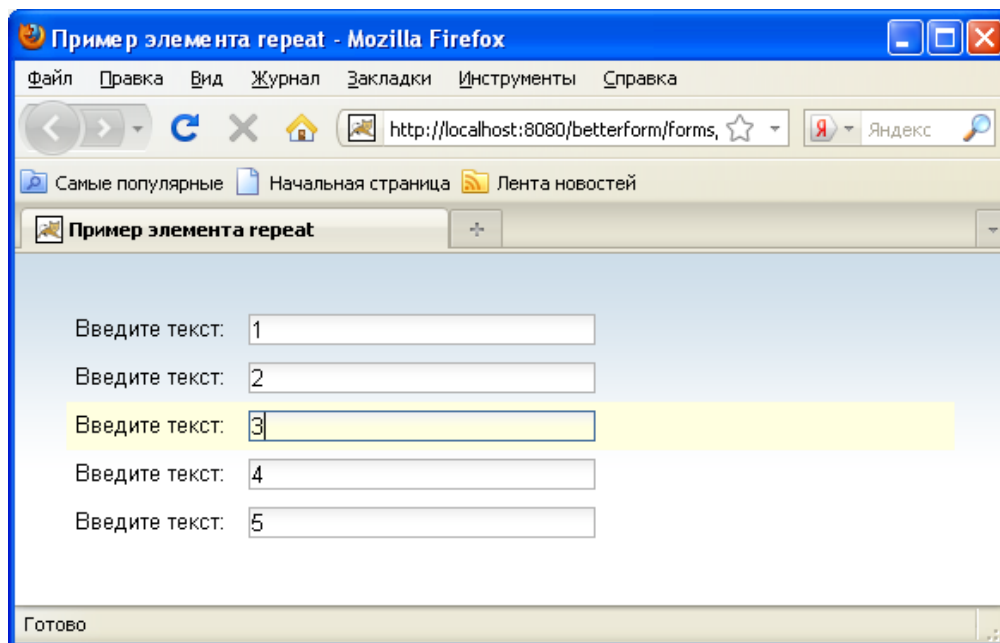


Рис. 5.23. Результат выполнения примера 5.3. Файл «repeat\_simple\_1.xhtml».

На элемент «repeat» оказывает влияние атрибут «appearance».

### Пример 5.3. Файл «repeat\_simple\_2.xhtml».

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример элемента repeat и атрибута appearance</title>

    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model>
      <xforms:instance>
        <example xmlns="">
          <item>
            <text>текст 1</text>
            <int>1</int>
          </item>
          <item>
            <text>текст 2</text>
```

```

        <int>2</int>
    </item>
    <item>
        <text>текст 3</text>
        <int>3</int>
    </item>
</example>
</xforms:instance>

<xforms:bind nodeset="int" type="xforms:integer"/>

</xforms:model>
</head>

<body>
    <h1>repeat (appearance="full")</h1>
    <xforms:repeat nodeset="item" appearance="full">
        <xforms:input ref="text">
            <xforms:label>Введите текст:</xforms:label>
        </xforms:input>
        <xforms:input ref="int">
            <xforms:label>Введите число:</xforms:label>
        </xforms:input>
    </xforms:repeat>

    <h1>repeat (appearance="compact")</h1>
    <xforms:repeat nodeset="item" appearance="compact">
        <xforms:input ref="text">
            <xforms:label>Введите текст:</xforms:label>
        </xforms:input>
        <xforms:input ref="int">
            <xforms:label>Введите число:</xforms:label>
        </xforms:input>
    </xforms:repeat>

```

```

<h1>repeat (appearance="minimal")</h1>
<xforms:repeat nodeset="item" appearance="minimal">
  <xforms:input ref="text">
    <xforms:label>Введите текст:</xforms:label>
  </xforms:input>
  <xforms:input ref="int">
    <xforms:label>Введите число:</xforms:label>
  </xforms:input>
</xforms:repeat>

</body>
</html>

```

Результат выполнения примера:

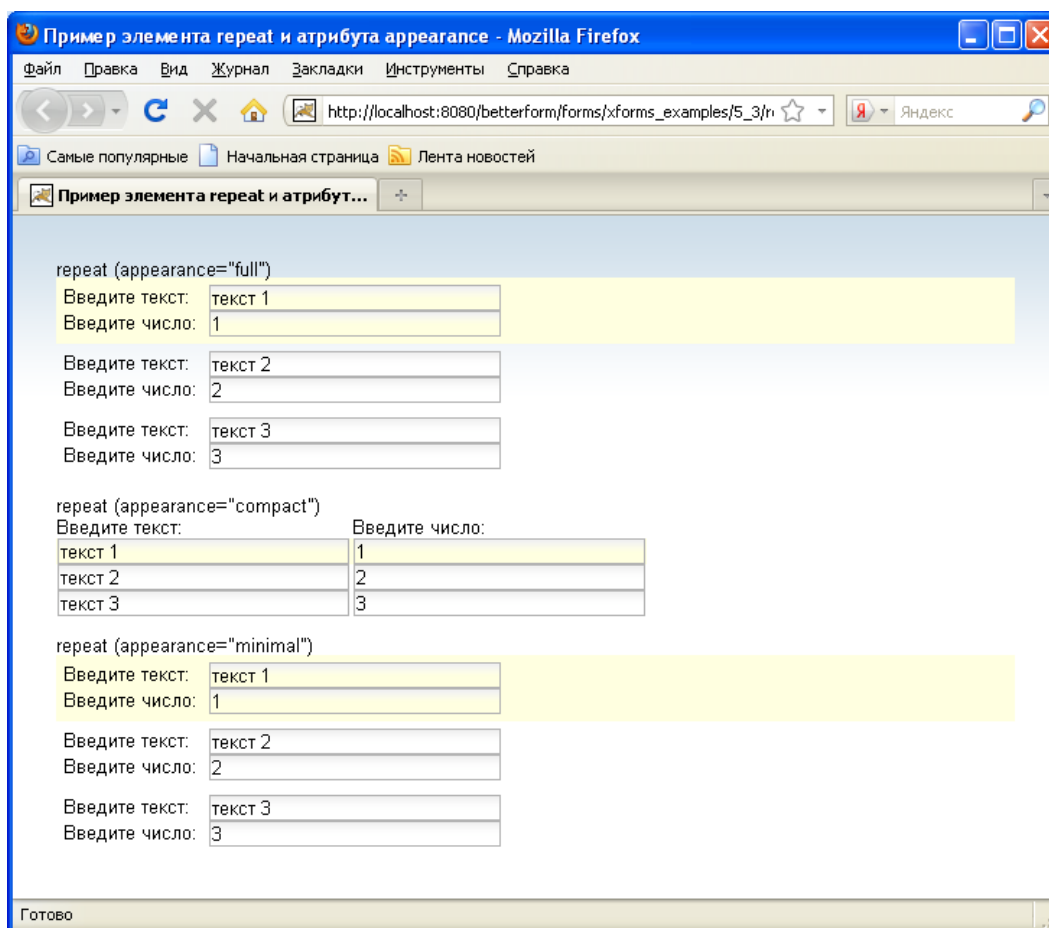


Рис. 5.24. Результат выполнения примера 5.3. Файл «repeat\_simple\_2.xhtml».

В следующем примере внутри элемента «xforms:repeat» используются элементы «xforms:switch».

Может показаться, что значения атрибутов id элементов «xforms:switch» перестают быть уникальными (так как «xforms:switch» повторяется в «xforms:repeat»), но в этом случае элементы «xforms:switch» работают корректно.

### **Пример 5.3. Файл «repeat\_switch.xhtml».**

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример элементов repeat и switch</title>

    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model id="result_model">
      <xforms:instance>
        <example xmlns="">
          <result>val1</result>
          <num>3</num>
        </example>
      </xforms:instance>
    </xforms:model>

    <xforms:model id="list_data">
      <xforms:instance>
        <example xmlns="">
          <item>
            <text>Значение 1</text>
          </item>
          <item>
            <text>Значение 2</text>
          </item>
          <item>
```

```

        <text>Значение 3</text>
    </item>
</example>
</xforms:instance>
</xforms:model>
</head>

<body>
    <xforms:repeat model="list_data" nodeset="item">

        <!-- Кнопки для переключения -->
        <xforms:group appearance="minimal">
            <xforms:trigger>
                <xforms:label>full</xforms:label>
                <xforms:toggle ev:event="DOMActivate" case="select_1"/>
            </xforms:trigger>

            <xforms:trigger>
                <xforms:label>compact</xforms:label>
                <xforms:toggle ev:event="DOMActivate" case="select_2"/>
            </xforms:trigger>

            <xforms:trigger>
                <xforms:label>minimal</xforms:label>
                <xforms:toggle ev:event="DOMActivate" case="select_3"/>
            </xforms:trigger>
        </xforms:group>

        <!-- В случае повторяющихся переключателей каждый
переключатель работает независимо от других -->
        <xforms:switch>
            <xforms:case id="select_1" selected="true">
                <xforms:output model="list_data" value="text"/>
                <xforms:trigger appearance="full">
                    <xforms:label model="list_data" ref="text"/>

```

```

    </xforms:trigger>
</xforms:case>

<xforms:case id="select_2">
  <xforms:output model="list_data" value="text"/>
  <xforms:trigger appearance="compact">
    <xforms:label model="list_data" ref="text"/>
  </xforms:trigger>
</xforms:case>

<xforms:case id="select_3">
  <xforms:output model="list_data" value="text"/>
  <xforms:trigger appearance="minimal">
    <xforms:label model="list_data" ref="text"/>
  </xforms:trigger>
</xforms:case>
</xforms:switch>
</xforms:repeat>
</body>
</html>

```

Результат выполнения примера:

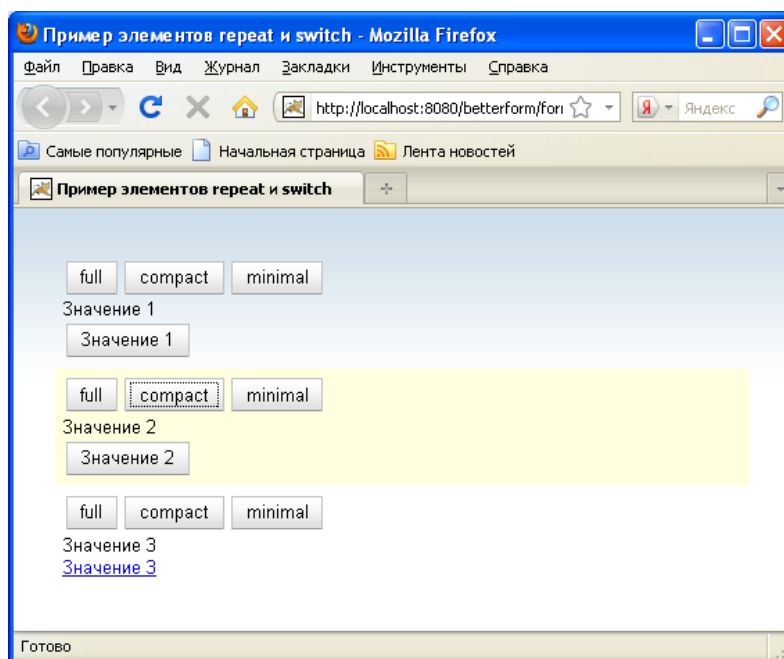


Рис. 5.25. Результат выполнения примера 5.3. Файл «repeat\_switch.xhtml».



Элементы «xforms:repeat» могут быть вложенными.

**Пример 5.3. Файл «repeat\_nested.xhtml».**

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример вложенных циклов repeat</title>
    <xforms:model>
      <xforms:instance>
        <example xmlns="">
          <item label="Группа 1">
            <text>Значение 11</text>
            <text>Значение 12</text>
            <text>Значение 13</text>
          </item>
          <item label="Группа 2">
            <text>Значение 21</text>
            <text>Значение 21</text>
          </item>
          <item label="Группа 3">
            <text>Значение 3</text>
          </item>
        </example>
      </xforms:instance>
    </xforms:model>
  </head>

  <body>
<!-- Внешний цикл перебирает элементы item и генерирует группы -->
    <xforms:repeat nodeset="item">
      <xforms:group appearance="minimal">
        <xforms:label ref="@label"/>
```

```

<!-- Вложенный цикл перебирает элементы text и генерирует поля ввода
-->

    <xforms:repeat nodeset="text">
        <xforms:input ref=".">
            <xforms:label>Введите текст:</xforms:label>
        </xforms:input>
    </xforms:repeat>
</xforms:group>
</xforms:repeat>
</body>
</html>

```

Результат выполнения примера:

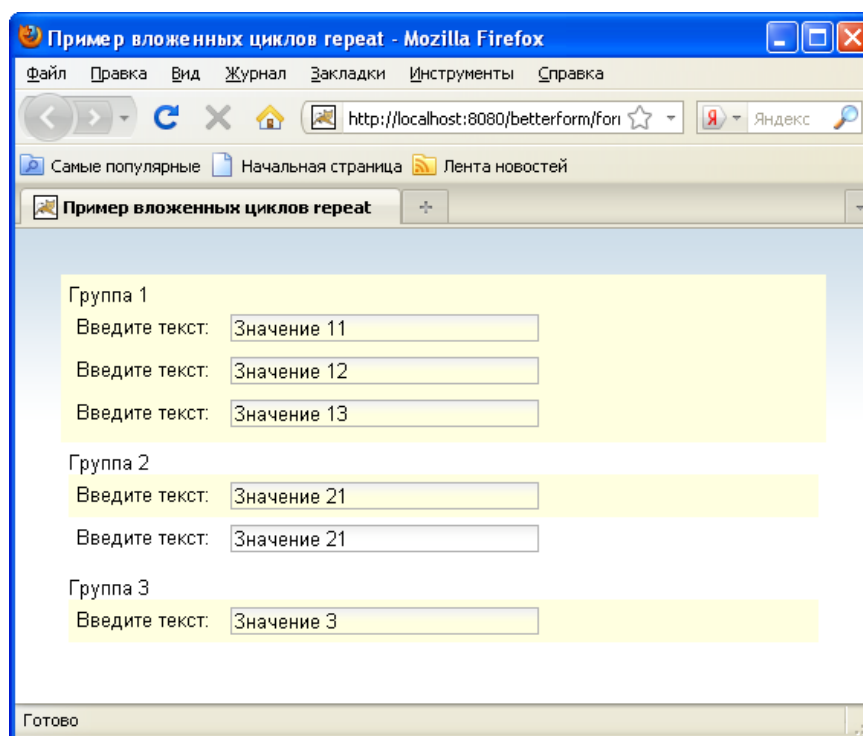


Рис. 5.26. Результат выполнения примера 5.3. Файл «repeat\_nested.xhtml».

Элемент «`xforms:repeat`» с вложенными элементами «`xforms:group`» позволяет реализовать ветвление на основе данных формы.

### Пример 5.3. Файл «repeat\_variant.xhtml».

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

```

<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Использование repeat с несколькими group</title>
    <xforms:model>
      <xforms:instance>
        <example xmlns="">
          <item123 type="type3">
            <date/>
            <datetime/>
          </item123>
          <item123 type="type2">
            <text>type2</text>
            <int>333</int>
          </item123>
          <item type="type1">
            <text>type1</text>
            <int>333</int>
          </item>
        </example>
      </xforms:instance>

      <xforms:bind nodeset="*/int" type="xforms:integer"/>
      <xforms:bind nodeset="*/date" type="xforms:date"/>
      <xforms:bind nodeset="*/datetime" type="xforms:dateTime"/>
    </xforms:model>
  </head>

  <body>

    <!-- Перебор всех элементов модели на 1 глубине вложенности, у
элементов могут быть различные названия -->
    <xforms:repeat nodeset="*">

```

**<!-- Группа соответствует конкретному перебираемому элементу, как найти нужный элемент? В атрибуте ref можно проверять его название, вложенные элементы, атрибуты. Внутри группы находятся элементы формы, специфичные для данного случая. Порядок следования групп не важен, группы будут идти в соответствии с тем порядком элементов в модели. -->**

```
<xforms:group ref=".[@type='type1' and name()='item']">
```

```
  <xforms:label ref="@type"/>
```

```
  <xforms:input ref="text">
```

```
    <xforms:label>Введите текст:</xforms:label>
```

```
</xforms:input>
```

```
  <xforms:input ref="int">
```

```
    <xforms:label>Введите число:</xforms:label>
```

```
</xforms:input>
```

```
</xforms:group>
```

**<!-- проверка того, что в текущий элемент вложены атрибут type='type2' и элементы text и int -->**

```
<xforms:group ref=".[@type='type2' and text and int]">
```

```
  <xforms:label ref="@type"/>
```

```
  <xforms:input ref="text">
```

```
    <xforms:label>Введите текст:</xforms:label>
```

```
</xforms:input>
```

```
  <xforms:input ref="int">
```

```
    <xforms:label>Введите число:</xforms:label>
```

```
</xforms:input>
```

```
  <xforms:output ref="int">
```

```
    <xforms:label>Число:</xforms:label>
```

```
</xforms:output>
```

```
</xforms:group>
```

```

<xforms:group ref=".[@type='type3']">
  <xforms:label ref="@type"/>

  <xforms:input ref="date">
    <xforms:label>Введите дату:</xforms:label>
  </xforms:input>

  <xforms:input ref="datetime">
    <xforms:label>Введите дату и время:</xforms:label>
  </xforms:input>
</xforms:group>
</xforms:repeat>
</body>
</html>

```

Результат выполнения примера:

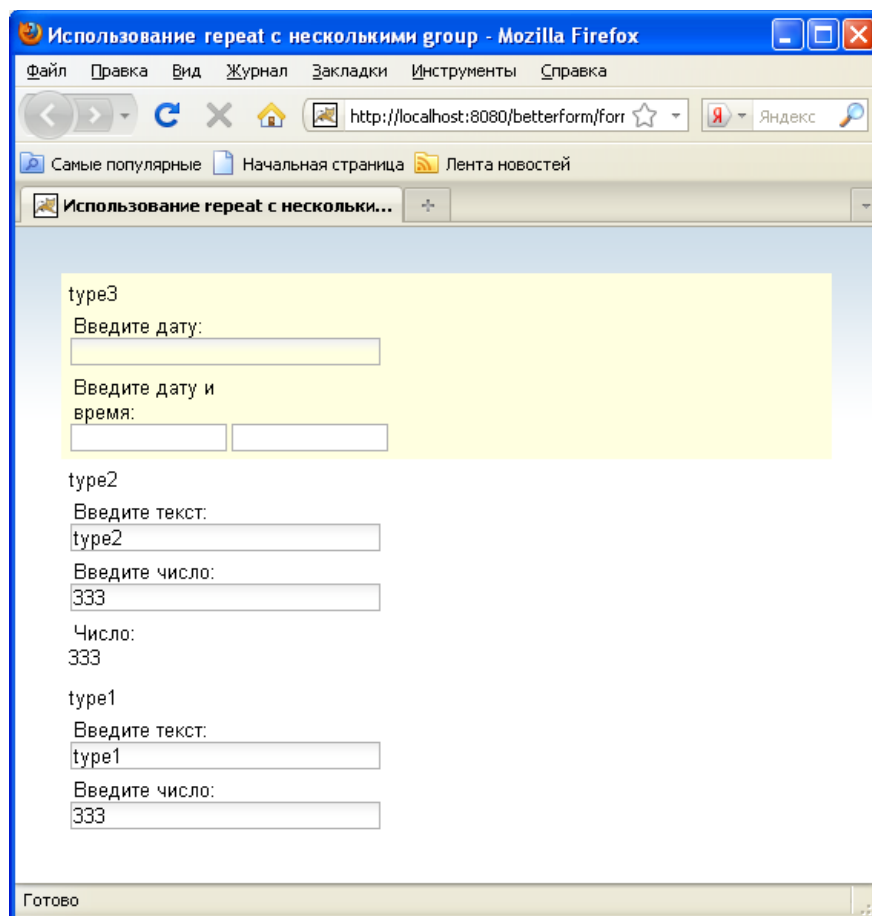


Рис. 5.27. Результат выполнения примера 5.3. Файл «repeat\_variant.xhtml».

У элемента «xforms:repeat» существует важное ограничение. Он может быть вложен не во все элементы XHTML, например, его нельзя использовать с табличными элементами. То есть из-за ограничений спецификации XHTML нельзя сделать таблицу, в которой строки повторялись бы с помощью элемента «xforms:repeat».

Для преодоления этого ограничения в спецификацию XForms были введены атрибуты, которые аналогичны атрибутам элемента «xforms:repeat»:

- repeat-model
- repeat-bind
- repeat-nodeset
- repeat-startindex
- repeat-number

Эти атрибуты должны быть атрибутами XHTML-элементов. То есть вместо использования конструкции (которая не соответствует стандарту XHTML):

```
<table>
  <repeat nodeset="...">
```

используется конструкция (которая соответствует стандарту XHTML):

```
<table xforms:repeat-nodeset="...">
```

В следующем примере для создания повторяющихся строк в таблице вместо элемента «xforms:repeat» используется атрибут «repeat-nodeset».

### **Пример 5.3. Файл «repeat\_table.xhtml».**

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример таблицы и элемента repeat</title>
    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model>
      <xforms:instance>
        <example xmlns="">
```

```

<item>
  <text>текст 1</text>
  <int>1</int>
</item>
<item>
  <text>текст 2</text>
  <int>2</int>
</item>
<item>
  <text>текст 3</text>
  <int>3</int>
</item>
</example>
</xforms:instance>

```

```

  <xforms:bind nodeset="int" type="xforms:integer"/>
</xforms:model>
</head>

```

```

<body>

```

**<!-- Атрибут `xforms:repeat-nodeset="item"` выполняет функцию элемента `repeat`. Элемент `repeat` (из схемы `xforms`) не может быть вложен в `table` (из схемы `xhtml`), так как спецификация `xhtml` этого не допускает -->**

```

  <table border="2" xforms:repeat-nodeset="item">
    <tr>
      <td><xforms:input ref="text"/></td>
      <td><xforms:input ref="int"/></td>
    </tr>
  </table>
</body>
</html>

```

Результат выполнения примера:

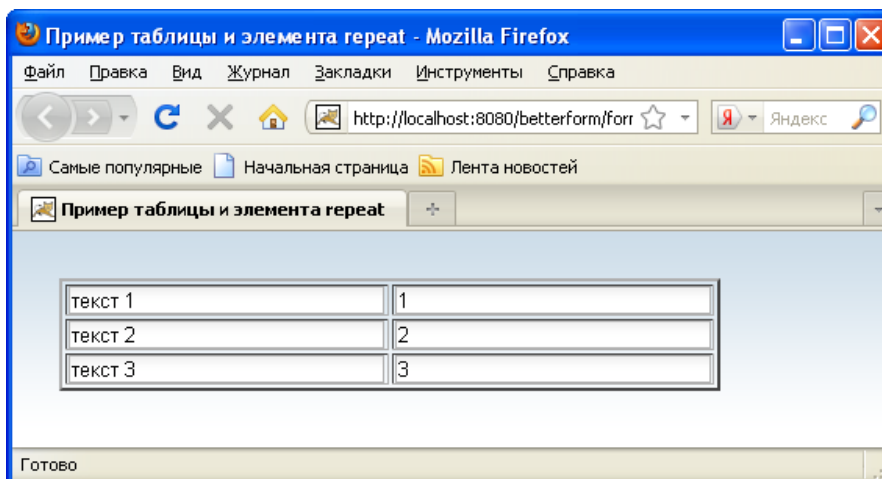


Рис. 5.28. Результат выполнения примера 5.3. Файл «repeat\_table.xhtml».

## 5.8 Элементы обработки событий

Элементы обработки событий позволяют обрабатывать события и менять XML-данные в модели данных.

Эти элементы не используются в простых XForms-формах. Они нужны для создания сложных интерактивных форм.

Мы будем использовать только основные элементы обработки событий.

### 5.8.1 Элемент action

Позволяет группировать другие элементы обработки событий.

Этот элемент удобно использовать как операторные скобки для условия или цикла.

Условие и цикл можно задавать с помощью атрибутов «if=XPath-выражение, возвращающее логическое значение» и «while=XPath-выражение, возвращающее логическое значение».

Если задан атрибут «if», то действие выполняется один раз, если логическое значение истинно.

Если задан атрибут «while», то действие выполняется многократно, пока логическое значение истинно.



Также обязательно должен быть задан атрибут «ev:event=событие», который определяет, обработчиком какого события является данный элемент. Префикс «ev» соответствует пространству имен событий XML (xmlns:ev="http://www.w3.org/2001/xml-events").

Полный список событий приведен в спецификации XForms. Наиболее часто используются следующие события:

- DOMActivate (событие по-умолчанию для элемента, например, нажатие на кнопку).
- DOMFocusIn xforms-focus (получение фокуса элементом управления).
- DOMFocusOut (потеря фокуса элементом управления).
- xforms-value-changed (изменение значения данных).
- xforms-valid (в результате изменения данных форма заполнена правильно).
- xforms-invalid (в результате изменения данных форма заполнена неправильно).

Эти атрибуты могут присутствовать у всех элементов обработки событий.

### 5.8.2 Элемент setvalue

Позволяет менять значение XML-элемента данных.

Ссылка на элемент данных задается с помощью атрибутов группы «Single-Node Binding» (ref, model, bind).

Новое значение задается в виде содержимого элемента «setvalue» или с помощью атрибута «value».

Могут быть заданы атрибуты «if» и «while».

Должен быть задан атрибут события «ev:event=событие». Атрибут события можно не задавать, только если элемент «setvalue» вложен в элемент «action», у которого задано событие.

Примеры:

```
<xforms:setvalue ev:event="DOMActivate" model="model1"
ref="text2">фиксированное значение</xforms:setvalue>
```

[Оглавление](#)

```
<xforms:setvalue ev:event="DOMActivate" model="model1" ref="text2"
value="concat(context()/text1, ' !!!') " />
```

### 5.8.3 Элемент delete

Позволяет удалять элемент данных из модели.

Атрибут «context=XPath-выражение» задает контекст, в котором удаляется элемент.

Атрибут «nodeset=XPath-выражение» позволяет указать название повторяющегося элемента для удаления. Это используется в случае удаления повторяющихся элементов в «xforms:repeat».

Атрибут «at=XPath-выражение» позволяет указать позицию удаляемого элемента.

Пример:

```
<xforms:delete ev:event="DOMActivate" model="model2"
context="repeat" nodeset="num" at="index('repeat1')"/>
```

Функция `index` возвращает номер выбранного элемента данных в «xforms:repeat», что позволяет удалять текущий (выбранный) элемент из «xforms:repeat».

### 5.8.4 Элемент insert

Позволяет добавлять элемент данных.

Атрибут «model» задает модель данных.

Следующие атрибуты задают приемник данных.

Атрибут «context» задает контекст, а атрибут «nodeset» указывает, на какой элемент нужно ориентироваться при добавлении.

Атрибут «at» указывает позицию добавления.

Атрибут «position= before | after» предписывает добавлять до или после заданной позиции.

Источник данных задается атрибутом «origin».

Пример:

```
<xforms:insert ev:event="DOMActivate" model="model2"
context="repeat" nodeset="num" at="index('repeat1')"
position="after" origin="instance('model2inst')/newvalue/num"/>
```

### 5.8.5 Элемент **setindex**

Устанавливает текущую позицию в элементе «xforms:repeat».

Атрибут «repeat» содержит ссылку на элемент «xforms:repeat», а атрибут «index» содержит номер устанавливаемой позиции.

Пример:

```
<xforms:setindex repeat="repeat1" index="1" />
```

### 5.8.6 Элемент **setfocus**

Устанавливает фокус на нужный элемент управления.

Уникальный id элемента управления задается в атрибуте «control» или вложенном элементе «control».

Пример:

```
<xforms:setfocus ev:event="DOMActivate" control="input_1" />

<xforms:setfocus ev:event="DOMActivate">
  <xforms:control value="concat('input_', '2')"/>
</xforms:setfocus>
```

### 5.8.7 Элемент **dispatch**

Позволяет направить событие элементу, id которого задается атрибутом «targetid». Атрибут «name» определяет название события, атрибут «delay» определяет временную задержку перед отправкой события.

С помощью этого элемента можно направлять как стандартные события, так и события, определяемые разработчиком.

Пример:

```
<xforms:dispatch name="xforms-refresh" targetid="model4" />
```

```
<xforms:dispatch name="new-event" targetid="model4" delay="5000" />
```

### 5.8.8 Элемент refresh

Обновление данных, относящихся к заданной модели. С помощью атрибута «model» можно задать модель данных.

### 5.8.9 Элемент load

Загрузка ресурса в окно браузера.

Атрибут «resource» задает URI для загрузки, атрибут «show = new | replace» позволяет загружать ресурс в новое или текущее окно браузера.

Пример:

```
<xforms:load ev:event="DOMActivate" show="new"
resource="input.xhtml"/>
```

### 5.8.10 Элемент send

Отправка данных формы на сервер из обработчика события. Атрибут «submission» ссылается на элемент «submission», который определяет параметры отправки формы на сервер.

Отличие «submit» состоит в том, что «submit» является элементом управления и отображается в виде кнопки или гиперссылки. Элемент «send» выполняет действие по отправке формы на сервер и не является элементом управления.

### 5.8.11 Элемент message

Используется для выдачи сообщения.

Может содержать атрибут «level= ephemeral | modeless | modal», который задает тип сообщения: ephemeral (всплывающая подсказка), modeless (немодальное окно), modal (модальное окно).

### 5.8.12 Примеры обработки событий

Следующий пример реализует основные возможности элементов обработки событий.

#### Пример 5.4. Файл «actions.xhtml».

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Примеры действий</title>

    <xforms:model id="model1">
      <xforms:instance>
        <example xmlns="">
          <text1>333</text1>
          <text2/>
        </example>
      </xforms:instance>
    </xforms:model>

    <xforms:model id="model2">
      <xforms:instance id="model2inst">
        <example xmlns="">
          <repeat>
            <num>1</num>
            <num>2</num>
            <num>3</num>
            <num>4</num>
            <num>5</num>
          </repeat>
          <newvalue>
            <num>1000</num>
          </newvalue>
          <newvalues>
```

```

        <num>100</num>
        <num>101</num>
        <num>102</num>
    </newvalues>
</example>
</xforms:instance>
</xforms:model>

```

```

<xforms:model id="model3">
    <xforms:instance>
        <example xmlns="">
            <uri>input.xhtml</uri>
        </example>
    </xforms:instance>
</xforms:model>

```

```

<xforms:model id="model4">
    <xforms:instance>
        <example xmlns="">
            <dt/>
        </example>
    </xforms:instance>

```

```

    <xforms:bind nodeset="dt" type="xforms:date"/>
    <!-- сообщения для реакции на события -->
    <xforms:message level="modal" ev:event="xforms-ready">Форма
загружена (сообщение появляется при загрузке формы)</xforms:message>
    <xforms:message level="modal" ev:event="xforms-refresh">Форма
обновлена</xforms:message>
    <xforms:message level="modal" ev:event="new-
event">Нестандартное событие</xforms:message>
</xforms:model>
</head>

<body>

```

```

<xforms:group>
  <xforms:label>Использование setvalue и setfocus</xforms:label>

  <xforms:input id="input_1" model="model1" ref="text1">
    <xforms:label>Введите текст:</xforms:label>
  </xforms:input>

  <xforms:input id="input_2" model="model1" ref="text2">
    <xforms:label>Установленное значение:</xforms:label>
  </xforms:input>

  <xforms:trigger>
    <xforms:label>Установить фиксированное
значение</xforms:label>
    <xforms:setvalue ev:event="DOMActivate" model="model1"
ref="text2">фиксированное значение</xforms:setvalue>
  </xforms:trigger>

  <xforms:trigger>
    <xforms:label>Установить введенное значение</xforms:label>
    <xforms:setvalue ev:event="DOMActivate" model="model1"
ref="text2" value="concat(context()/text1, ' !!!') " />
  </xforms:trigger>

  <xforms:trigger>
    <xforms:label>Установить фокус на первое поле
ввода</xforms:label>
    <xforms:setfocus ev:event="DOMActivate" control="input_1" />
  </xforms:trigger>

  <xforms:trigger>
    <xforms:label>Установить фокус на второе поле
ввода</xforms:label>
    <xforms:setfocus ev:event="DOMActivate">
      <!-- Используется вложенный элемент control -->

```

```

        <xforms:control value="concat('input_', '2')"/>
    </xforms:setfocus>
</xforms:trigger>
</xforms:group>

<hr/>

<xforms:group>
    <xforms:label>Добавление элементов в
последовательность</xforms:label>

    <!-- последовательность элементов -->
    <xforms:repeat id="repeat1" model="model2"
nodeset="repeat/num">
        <xforms:input model="model2" ref="."/>
        <xforms:refresh />
    </xforms:repeat>

    <xforms:output model="model2" value="count(repeat/num)">
        <xforms:label>Количество элементов:</xforms:label>
    </xforms:output>

    <xforms:input model="model2" ref="newvalue/num">
        <xforms:label>Добавляемый элемент:</xforms:label>
    </xforms:input>

    <xforms:trigger>
        <xforms:label>Добавить перед выбранным</xforms:label>
        <xforms:insert ev:event="DOMActivate" model="model2"
context="repeat" nodeset="num" at="index('repeat1') "
position="before" origin="instance('model2inst')/newvalue/num"/>
    </xforms:trigger>

    <xforms:trigger>
        <xforms:label>Добавить после выбранного</xforms:label>

```



```

    <xforms:insert ev:event="DOMActivate" model="model2"
context="repeat" nodeset="num" at="index('repeat1') "
position="after" origin="instance('model2inst')/newvalue/num"/>
  </xforms:trigger>

```

```

  <xforms:trigger>
    <xforms:label>Добавить несколько элементов после
выбранного</xforms:label>
    <!-- Добавляются все элементы num, вложенные в newvalues -->
    <xforms:insert ev:event="DOMActivate" model="model2"
context="repeat" nodeset="num" at="index('repeat1') "
position="after" origin="instance('model2inst')/newvalues/num"/>
  </xforms:trigger>

```

```

  <xforms:trigger>
    <xforms:label>Добавить последний после
выбранного</xforms:label>
    <xforms:insert ev:event="DOMActivate" model="model2"
context="repeat" nodeset="num" at="index('repeat1')"/>
  </xforms:trigger>

```

```

  <xforms:trigger>
    <xforms:label>Продублировать последний</xforms:label>
    <!-- Если не указан исходный элемент для добавления, то в
качестве исходного используется последний элемент в
последовательности -->
    <!-- Если не указано место для добавления, то добавление
производится после последнего элемента -->
    <xforms:insert ev:event="DOMActivate" model="model2"
context="repeat" nodeset="num"/>
  </xforms:trigger>

```

```

  <xforms:trigger>

```

```

    <xforms:label>Добавить константу после
выбранного</xforms:label>

    <!-- Элемент insert используется для вставки в
последовательность, а элемент setvalue для установки значения по
умолчанию -->

    <xforms:insert ev:event="DOMActivate" model="model2"
context="repeat" nodeset="num" at="index('repeat1')"
position="after" origin="instance('model2inst')/newvalue/num"/>

    <!-- После добавления добавленный элемент становится текущим
и функция index возвращает его номер -->

    <xforms:setvalue ev:event="DOMActivate" model="model2"
ref="repeat/num[index('repeat1')]">фиксированное
значение</xforms:setvalue>

    </xforms:trigger>

<xforms:trigger>
    <xforms:label>Добавить константу после выбранного
2</xforms:label>

    <!-- несколько действий могут быть сгруппированы в элемент
action -->

    <xforms:action ev:event="DOMActivate">

        <!-- Если шаблон для добавления в атрибуте origin незадан,
то шаблоном является последний элемент списка -->

        <xforms:insert model="model2" context="repeat"
nodeset="num" at="index('repeat1')" position="after" />

        <!-- После добавления добавленный элемент становится
текущим и функция index возвращает его номер -->

        <xforms:setvalue model="model2"
ref="repeat/num[index('repeat1')]">фиксированное
значение</xforms:setvalue>

    </xforms:action>
</xforms:trigger>

<xforms:trigger>

```

```

    <xforms:label>Удалить выбранный</xforms:label>
    <xforms:delete ev:event="DOMActivate" model="model2"
context="repeat" nodeset="num" at="index('repeat1')"/>
  </xforms:trigger>

  <xforms:trigger>
    <xforms:label>Очистить список</xforms:label>
    <xforms:delete ev:event="DOMActivate" model="model2"
context="repeat" nodeset="*"/>
  </xforms:trigger>

  <xforms:trigger>
    <xforms:label>Пронумеровать элементы</xforms:label>
    <!-- несколько действий могут быть сгруппированы в элемент
action -->
    <xforms:action ev:event="DOMActivate">
      <!-- Установка позиции по умолчанию в 1 -->
      <xforms:setindex repeat="repeat1" index="1" />

      <!-- Элемент action выполняется с условием, пока номер
текущего элемента в repeat <= 5 (вместо константы должна быть
проверка количества элементов) -->
      <!-- Должно быть XPath-выражение index('repeat1') &lt;=
count(instance('model2inst')/repeat/num) -->
      <!-- но сравнение с количеством элементов в правой части
не работает -->
      <!--поэтому вместо
count(instance('model2inst')/repeat/num) используется константа 5 --
>
      <!-- index('repeat1') возвращает текущую позицию в repeat
-->
      <xforms:action while="index('repeat1') &lt;= 5">
        <!-- Элемент setvalue присваивает текущему элементу
repeat/num (в квадратных скобках указывается индекс текущего
элемента) позицию этого элемента в repeat -->

```

```

        <xforms:setvalue model="model2"
ref="repeat/num[index('repeat1')]" value="index('repeat1')"/>
        <!-- Переход к следующему элементу repeat -->
        <xforms:setindex repeat="repeat1"
index="index('repeat1') + 1" />
    </xforms:action>
</xforms:action>
</xforms:trigger>

<xforms:trigger>
    <xforms:label>Отмена изменений в форме</xforms:label>
    <xforms:reset ev:event="DOMActivate" model="model2"/>
</xforms:trigger>

</xforms:group>

<hr/>

<xforms:group>
    <xforms:label>Навигация между формами</xforms:label>

    <xforms:trigger>
        <xforms:label>Открытие формы 1</xforms:label>
        <!-- Адрес читается из модели -->
        <xforms:load ev:event="DOMActivate" model="model3" ref="uri"
show="new"/>
    </xforms:trigger>

    <xforms:trigger>
        <xforms:label>Открытие формы 2</xforms:label>
        <!-- Адрес задан в виде константы с помощью атрибута
resource -->
        <xforms:load ev:event="DOMActivate" show="new"
resource="input.xhtml"/>
    </xforms:trigger>

```

```

    <xforms:trigger>
      <xforms:label>Открытие формы в текущем окне</xforms:label>
      <!-- Адрес задан в виде константы с помощью элемента
resource -->
      <xforms:load ev:event="DOMActivate" model="model3" ref="uri"
show="replace"/>
    </xforms:trigger>

  </xforms:group>

  <hr/>

  <xforms:group>
    <xforms:label>Обработка событий</xforms:label>

    <xforms:input model="model4" ref="dt">
      <xforms:label>Введите неправильную дату:</xforms:label>
      <xforms:message ev:event="xforms-invalid"><xforms:output
ref="."/> - неправильное значение для даты</xforms:message>
    </xforms:input>

    <xforms:trigger>
      <xforms:label>Отправка события refresh</xforms:label>
      <!-- Адрес задан в виде константы с помощью атрибута
resource -->
      <xforms:dispatch name="xforms-refresh" targetid="model4" />
    </xforms:trigger>

    <xforms:trigger>
      <xforms:label>Отправка собственного события с задержкой 5
секунд</xforms:label>
      <!-- Адрес задан в виде константы с помощью атрибута
resource -->

```

```

    <xforms:dispatch name="new-event" targetid="model4"
delay="5000" />
  </xforms:trigger>

</xforms:group>
</body>
</html>

```

Результат выполнения примера:

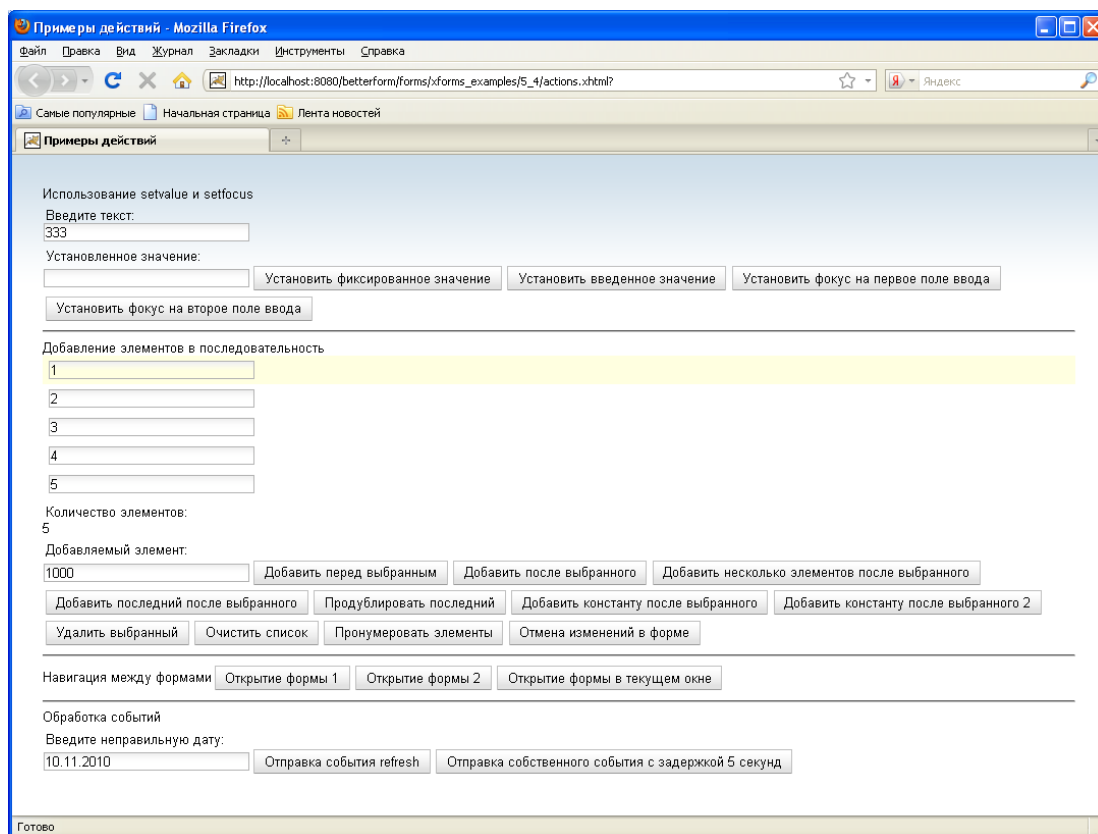


Рис. 5.29. Результат выполнения примера 5.4. Файл «actions.xhtml».

В следующем примере форма содержит фильтр по названию элемента.

#### Пример 5.4. Файл «filter.xhtml».

```

<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <head>
    <title>Пример фильтра</title>

```

[Оглавление](#)

```

<xforms:model>
  <xforms:instance>
    <example xmlns="">
      <item>
        <text>Пример гиперссылки 1</text>
        <uri>http://iu5.bmstu.ru/1.html</uri>
      </item>
      <item>
        <text>ПРИМЕР гиперссылки 12</text>
        <uri>http://iu5.bmstu.ru/2.html</uri>
      </item>
      <item>
        <text>ПриМер гиперссылки 123</text>
        <uri>http://iu5.bmstu.ru/3.html</uri>
      </item>
      <filter>123</filter>
    </example>
  </xforms:instance>

  <xforms:bind nodeset = "/example/item/uri" type="xsd:anyURI" />
</xforms:model>
</head>

<body>
  <xforms:group>

    <xforms:input ref="filter">
      <xforms:label>Введите текст фильтра:</xforms:label>
    </xforms:input>

    <!-- Выбираются только те элементы item, элементы text которых
содержат filter в качестве подстроки. nodeset=
"item[contains(text, //filter)]" - фильтр с учетом регистра символов,

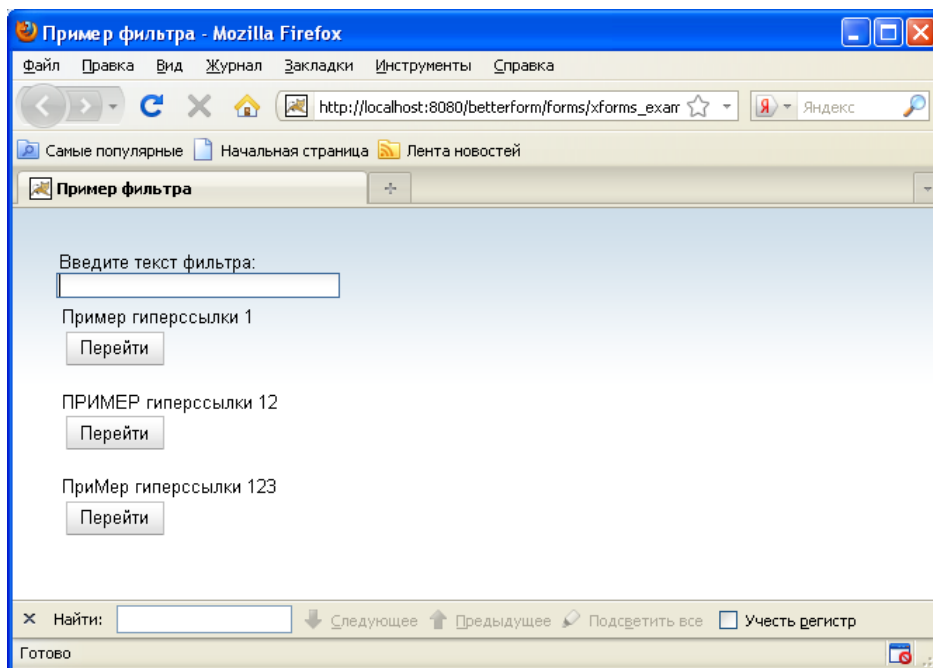
```

обычно это нежелательно. Поэтому в примере реализован Фильтр без учета регистра символов. -->

```
<xforms:repeat id="repeat1"
nodeset="item[contains(translate(text,'абвгдежзийклмнопрстуфхцчщъьэ
юя','АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЬЭЮЯ'),translate(//filter,'абвгдежзи
йклмнопрстуфхцчщъьэюя','АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЬЭЮЯ'))]">
    <xforms:output value="text"/>

    <xforms:trigger>
        <xforms:label>Перейти</xforms:label>
        <xforms:load ev:event="DOMActivate" show="new">
            <xforms:resource value="uri"/>
        </xforms:load>
    </xforms:trigger>
</xforms:repeat>
</xforms:group>
</body>
</html>
```

Результат выполнения примера:





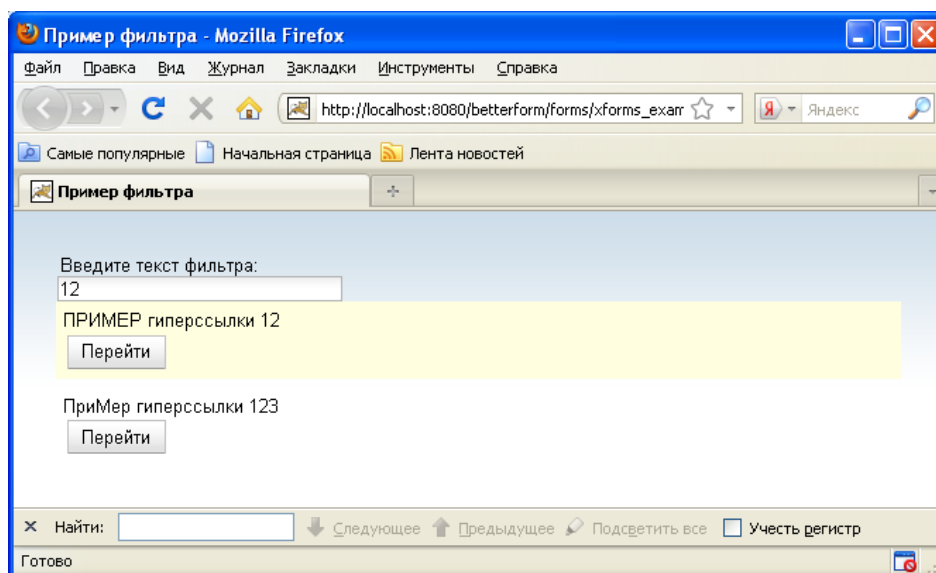


Рис. 5.30. Результат выполнения примера 5.4. Файл «filter.xhtml».

## 5.9 Сохранение данных формы

После заполнения полей формы, данные должны быть отправлены на сервер.

Параметры отправки данных определяются элементом «submission» в модели данных формы. На элемент «submission» могут ссылаться элементы «submit» (кнопка отправки) и «send» (отправка из обработчика события).

С помощью элемента «submission» данные могут быть переданы для обработки серверному сценарию или сохранены в СУБД «eXist».

Рассмотрим более подробно основные атрибуты элемента «submission».

Атрибут «id» определяет id элемента для дальнейших ссылок в форме.

Атрибут «action» задает URI серверного сценария, которому будут передаваться данные. Считается устаревшим, вместо него должен использоваться атрибут «resource».

Атрибут «ref=XPath-выражение» определяет, какую часть модели данных нужно отправлять, по умолчанию «/». Этот атрибут позволяет отправлять фрагмент модели данных.

Атрибут «method» задает метод HTTP-протокола для отправки данных (обычно используется POST, иногда PUT). Метод GET здесь не может быть использован, так как XML-данные не должны передаваться через строку URI.

[Оглавление](#)

Атрибут «`includenamespacesprefixes`» позволяет включать в передаваемые данные нужные префиксы пространств имен. Это полезно в том случае, когда данные содержат пространства имен. Пустое значение атрибута указывает, что префиксы пространств имен не передаются на сервер.

Атрибут «`replace`» определяет какую часть XForms-формы нужно заменить ответом, полученным от серверного сценария: «`all`» (полностью заменить форму), «`instance`» (заменить только данные формы), «`none`» (ничего не заменять).

В этом разделе для отправки мы будем использовать метод PUT и изучать отправленные файлы в «eXist». Для доступа к «eXist» через URI (REST-доступ) используется следующий URI: «`http://localhost:8080/exist/rest/db/название коллекции и имя файла.xml`».

В следующем примере заполненная форма сохраняется в «eXist» с помощью метода PUT протокола HTTP. Используется URI «`http://localhost:8080/exist/rest/db/exist_put.xml`», то есть данные сохраняются в файл «`exist_put.xml`» в корневой коллекции «`db`».

#### **Пример 5.5. Файл «`exist_put.xhtml`».**

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример сохранения данных с использованием метода
put</title>

    <!-- Пустой префикс пространства имен обязателен для корневого
элемента xforms:instance -->
    <xforms:model>
      <xforms:instance>
        <example xmlns="">
          <text/>
          <int/>
          <bool/>
```

```

    <date/>
    <time/>
    <datetime/>
  </example>
</xforms:instance>

```

```

<xforms:bind nodeset="bool" type="xforms:boolean"/>
<xforms:bind nodeset="int" type="xforms:integer"/>
<xforms:bind nodeset="date" type="xforms:date"/>
<xforms:bind nodeset="time" type="xforms:time"/>
<xforms:bind nodeset="datetime" type="xforms:dateTime"/>

```

```

    <xforms:submission replace="none"
action="http://localhost:8080/exist/rest/db/exist_put.xml"
method="put" id="submit_id"/>
  </xforms:model>
</head>

```

```

<body>
  <xforms:group>
    <xforms:input ref="text">
      <xforms:label>Введите текст:</xforms:label>
    </xforms:input>

    <xforms:input ref="int">
      <xforms:label>Введите число:</xforms:label>
    </xforms:input>

    <xforms:input ref="bool">
      <xforms:label>Введите логическое значение:</xforms:label>
    </xforms:input>

    <xforms:input ref="date">
      <xforms:label>Введите дату:</xforms:label>
    </xforms:input>

```

```

<xforms:input ref="time">
  <xforms:label>Введите время:</xforms:label>
</xforms:input>

<xforms:input ref="datetime">
  <xforms:label>Введите дату и время:</xforms:label>
</xforms:input>

<!-- Кнопка отправки формы. Атрибут submission ссылается на
элемент xforms:submission в модели -->
<xforms:submit submission="submit_id">
  <xforms:label>Отправка формы</xforms:label>
</xforms:submit>

</xforms:group>
</body>
</html>

```

Результат выполнения примера:

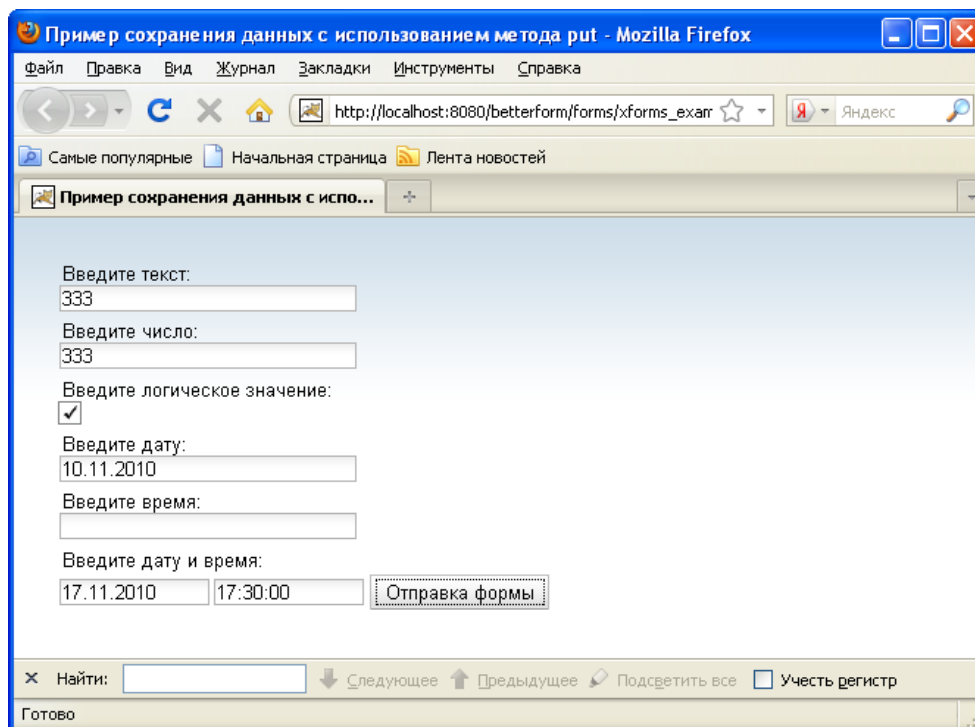


Рис. 5.31. Результат выполнения примера 5.5. Файл «exist\_put.xhtml».

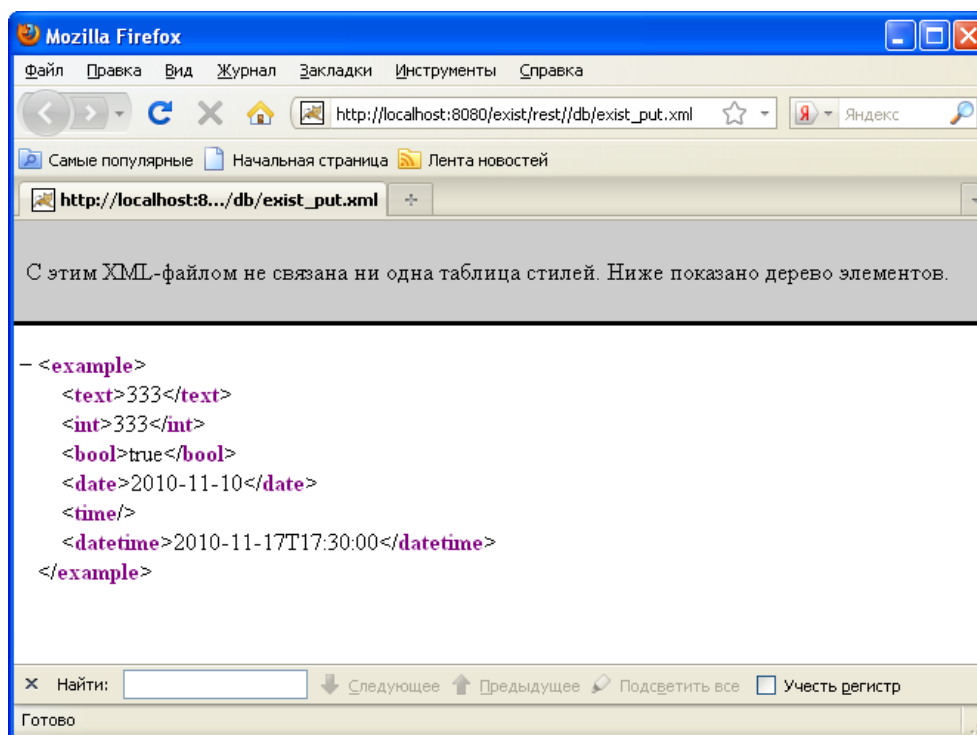


Рис. 5.32. Результат выполнения примера 5.5. Файл «exist\_put.xml».

В следующем примере используются пространства имен.

#### Пример 5.5. Файл «exist\_ns.xhtml».

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:iu5="http://iu5.bmstu.ru"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events" >
  <head>
    <title>Пример сохранения данных с использованием пространства
имен</title>
    <xforms:model>
      <xforms:instance>
        <iu5:example>
          <iu5:text/>
          <iu5:int/>
          <iu5:bool/>
          <iu5:time/>
          <iu5:datetime/>
          <!-- Пример вложенного элемента -->
```

[Оглавление](#)

```

    <iu5:container1>
      <iu5:container2>
        <iu5:date/>
      </iu5:container2>
    </iu5:container1>
  </iu5:example>
</xforms:instance>

<xforms:bind nodeset="iu5:bool" type="xforms:boolean"/>
<xforms:bind nodeset="iu5:int" type="xforms:integer"/>
<xforms:bind nodeset="iu5:container1/iu5:container2/iu5:date"
type="xforms:date"/>
<xforms:bind nodeset="iu5:time" type="xforms:time"/>
<xforms:bind nodeset="iu5:datetime" type="xforms:dateTime"/>

<xforms:submission replace="none"
action="http://localhost:8080/exist/rest/db/exist_ns.xml"
method="put" id="submit_id" includenamespaces="iu5" />
</xforms:model>
</head>

<body>
  <xforms:group>

    <xforms:input ref="iu5:text">
      <xforms:label>Введите текст:</xforms:label>
    </xforms:input>

    <xforms:input ref="iu5:int">
      <xforms:label>Введите число:</xforms:label>
    </xforms:input>

    <xforms:input ref="iu5:bool">
      <xforms:label>Введите логическое значение:</xforms:label>
    </xforms:input>

```

```

<xforms:input ref="iu5:container1/iu5:container2/iu5:date">
  <xforms:label>Введите дату:</xforms:label>
</xforms:input>

<xforms:input ref="iu5:time">
  <xforms:label>Введите время:</xforms:label>
</xforms:input>

<xforms:input ref="iu5:datetime">
  <xforms:label>Введите дату и время:</xforms:label>
</xforms:input>

<!-- Кнопка отправки формы. Атрибут submission ссылается на
элемент xforms:submission в модели -->
<xforms:submit submission="submit_id">
  <xforms:label>Отправка формы</xforms:label>
</xforms:submit>

</xforms:group>
</body>
</html>

```

**Результат выполнения примера:**

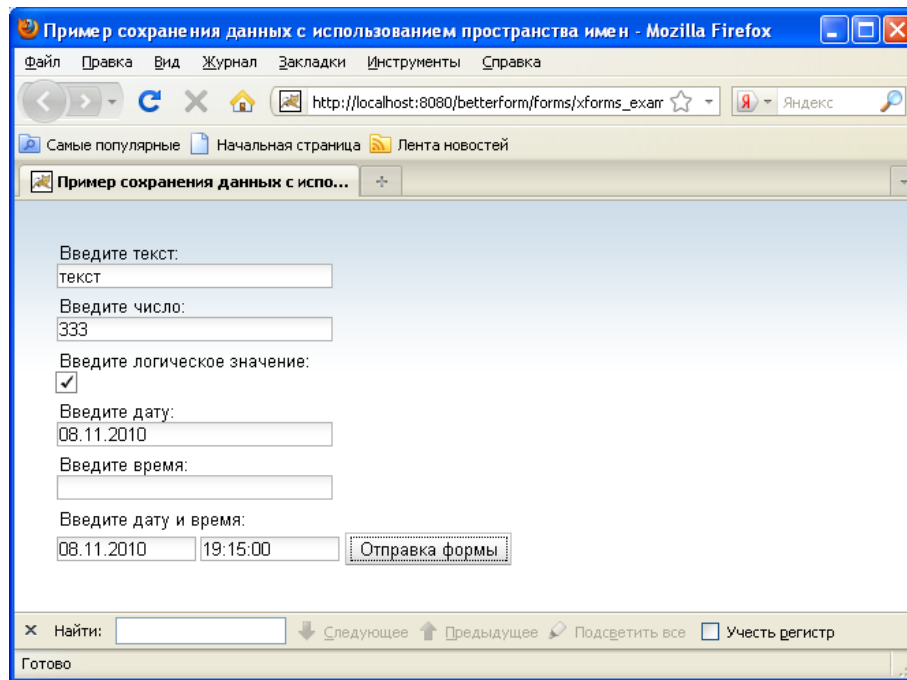


Рис. 5.33. Результат выполнения примера 5.5. Файл «exist\_ns.xhtml».

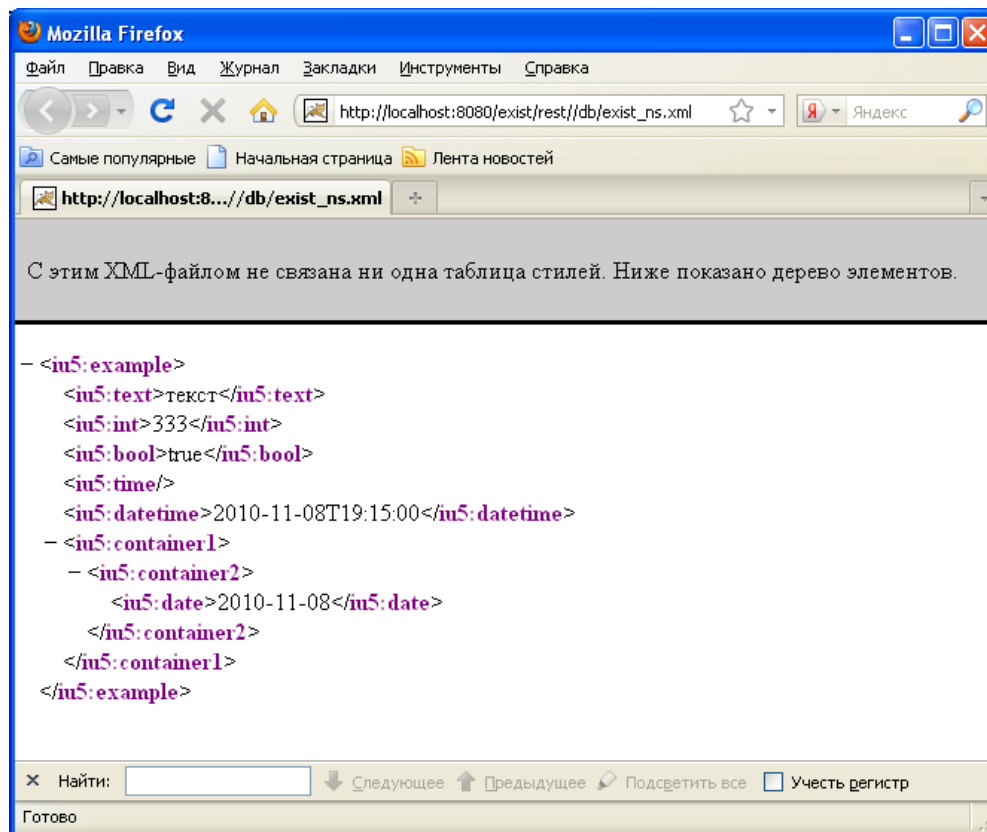


Рис. 5.34. Результат выполнения примера 5.5. Файл «exist\_ns.xml».

Более сложные примеры совместного использования XForms-форм и СУБД «eXist» будут рассмотрены в разделе XRX-приложений.



## 5.10 Использование таблиц стилей CSS в XForms-формах

При разработке форм XForms часто возникает необходимость изменить внешний вид элементов управления формы с помощью стилей CSS.

Особенностью является то, что XForms-процессоры преобразуют XHTML-документ (содержащий XForms-форму) в HTML-документ (обычно содержащий JavaScript и собственные стили CSS), поэтому не всегда понятно, к чему же нужно применять CSS-стиль.

Как правило, в каждом XForms-процессоре существует собственный способ применения стилей. Наиболее распространенными являются два способа:

1. Стили применяются к XForms-форме в исходном XHTML-документе. XForms-процессор учитывает эти стили при преобразовании формы в HTML.
2. Стили применяются к результирующему HTML-документу.

### 5.10.1 Применение стилей CSS к XForms-форме

В 1 способе CSS-стили применяются к XForms-элементам. Этот способ является рекомендуемым и в черновом варианте описан в стандарте XForms (в дальнейшем предполагается сделать это отдельным стандартом).

Но пока это черновая рекомендация, и ее только начинают поддерживать некоторые XForms-процессоры. В частности, этот способ используется в XSLTForms, и мы в дальнейшем будем применять его в XRX-приложениях.

Пример:

```
/* Объявление пространства имен xforms */
@namespace xf url('http://www.w3.org/2002/xforms');
/* Определение стиля для элемента input с псевдоклассом read-only */
xf|input { display: inline; }
```

В спецификации также определены дополнительные псевдоклассы и псевдоэлементы, предназначенные для работы с XForms, например псевдоклассы «:read-only» для обозначения элементов только для чтения и «:read-write» для остальных элементов.

### **5.10.2 Применение стилей CSS к результирующему HTML-документу**

Во 2 способе стили должны применяться к результирующему HTML-документу.

Но как узнать соответствие между исходными XForms-элементами и полученными после преобразования HTML-элементами?

В различных XForms-процессорах эта задача решается по-разному. Например, betterFORM добавляет при преобразовании специальные CSS-классы, которые «похожи» по названию на исходные XForms-элементы. Для этих CSS-классов можно задавать необходимые стилевые свойства. Это соответствие описано в документации к betterFORM в специальном документе «betterForm CSS Reference».

Предыдущий пример может быть записан для betterFORM следующим образом:

```
xfInput { display: inline; }
```

## **5.11 Материалы для дальнейшего изучения**

Рекомендуется ознакомиться со стандартом XForms [XForms, 2009].

Также можно ознакомиться со стандартными примерами и стандартной документацией betterFORM.

## **5.12 Контрольные вопросы**

1. Для чего предназначена технология XForms?
2. В чем основные отличия XForms-форм от HTML-форм?
3. Что означают понятия основной документ (host document) и основной язык разметки (host language)?
4. Что такое XForms-процессор? Каковы основные особенности XForms-процессоров?

5. В чем состоит принцип «конкретной» модели и «абстрактных» элементов управления формы?
6. Что может содержать модель данных формы?
7. Какие способы существуют для связи данных формы, ограничений и элементов управления формы?
8. Как используются типы данных при задании ограничений в XForms?
9. Как можно задавать ограничения на данные с помощью элемента `xforms:bind`?
10. Каким образом можно использовать несколько моделей и экземпляров в форме?
11. Какие основные элементы управления используются в XForms?
12. Как используются элементы обработки событий в XForms?
13. Как сохранить данные формы в СУБД «eXist»?
14. Как использовать таблицы стилей CSS в XForms-формах?

## 6 Часть 6. Технологии XInclude, XLink, XPointer

### 6.1 Технология XInclude

Технология XInclude [XInclude, 2006] позволяет осуществлять вставку одного XML-документа (или его фрагмента) в другой XML-документ.

Оба документа являются статическими. Использовать XQuery для динамической генерации документа не обязательно (хотя это возможно).

Технология XInclude частично поддерживается в СУБД «eXist».

Существуют еще две технологии, которые достаточно близки по назначению к технологии XInclude – это технологии XLink и XPointer.

### 6.2 Технология XLink

Технология XLink [XLink, 2010] позволяет описывать ссылки между XML-ресурсами. В отличие от простых однонаправленных ссылок HTML, XLink позволяет описывать двунаправленные, цепочные и другие виды ссылок.

XLink позволяет только описать ссылки между XML-ресурсами, но не говорит о том, как могут использоваться эти описания.

Эта спецификация должна прийти на смену простым однонаправленным ссылкам HTML, но пока она не очень широко используется. «eXist» не поддерживает эту спецификацию.

### 6.3 Технология XPointer

Технология XPointer позволяет выбирать фрагмент данных из XML-документа. Особенностью технологии XPointer является то, что она позволяет выбирать данные из XML-документа различными способами, эти способы называются схемами. Основная часть технологии XPointer (framework) рассмотрена в [XPointer Framework, 2003].

Схема `xpointer()` [**XPointer xpointer() Scheme, 2002**] позволяет выбирать данные на основе XPath-выражений. Именно эту схему мы будем использовать в дальнейших примерах.

Схема `xmlns()` [**XPointer xmlns() Scheme, 2003**] позволяет использовать в XPointer-запросах пространства имен.

Схема `element()` [**XPointer element() Scheme, 2003**] позволяет выбирать данные на основе отдельного языка запросов (он несколько похож на упрощенный XPath). Как правило, вместо этой схемы используется схема `xpointer()`.

## **6.4 Использование технологий XInclude и XPointer в eXist**

Для работы с примером 6.1 необходимо создать в «eXist» коллекцию «/db/iu5/include» и загрузить в нее все файлы примера 6.1.

Для запуска следующих примеров необходимо открыть eXide и выполнить следующий запрос:

```
let $doc := doc("/db/iu5/include/НАЗВАНИЕ_ФАЙЛА_ПРИМЕРА.xml")
return $doc
```

Далее в примерах используется файл «inner.xml». Этот документ целиком или частично будет добавляться в другие документы.

### **Файл inner.xml:**

```
<?xml version="1.0" encoding="utf-8"?>
<InnerDocument>
  <p>Это вложенный документ</p>
  <p id="id_inner">Это вложенный документ, id элемента=id_inner </p>
  <p>Это также вложенный документ</p>
</InnerDocument>
```

### 6.4.1 Элемент `xi:include`

Для работы с технологией XInclude используется пространство имен `"http://www.w3.org/2001/XInclude"`. Как правило, используется префикс пространства имен `"xi"`.

Основным элементом спецификации XInclude является элемент `xi:include`.

Если в текущем документе встречается команда `<xi:include href="вложенный документ.xml"/>`, то вместо `xi:include` добавляется содержимое вложенного документа.

#### Файл `include_1.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<Root xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>Следующий фрагмент добавлен с помощью XInclude:</p>
  <xi:include href="inner.xml"/>
</Root>
```

#### Результат выполнения:

```
<Root xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>Следующий фрагмент добавлен с помощью XInclude:</p>
  <InnerDocument>
    <p>Это вложенный документ</p>
    <p id="id_inner">Это вложенный документ, id
элемента=id_inner </p>
    <p>Это также вложенный документ</p>
  </InnerDocument>
</Root>
```

То есть элемент `<<xi:include href="inner.xml"/>>` был заменен файлом `inner.xml`.

В нашем случае в атрибуте `<href>` указано только название файла (файл расположен в той же коллекции). Однако, атрибут `<href>` может содержать также названия коллекций или даже произвольный URI (который начинается с `http://...`). Главное, чтобы был указан путь к документу в формате XML.

В следующем примере указан путь к документу в коллекции БД.

**Файл include\_2.xml:**

```
<?xml version="1.0" encoding="utf-8"?>
<Root xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>Следующий фрагмент добавлен с помощью XInclude:</p>
  <xi:include href="/db/iu5/include/inner.xml"/>
</Root>
```

Результат выполнения совпадает с предыдущим.

**6.4.2 Элемент xi:fallback**

Этот элемент используется для обработки ошибок, позволяет добавлять в документ фрагмент, если не был выполнен xi:include.

**Файл fallback\_1.xml:**

```
<?xml version="1.0" encoding="utf-8"?>
<Root xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>Следующий фрагмент добавлен с помощью XInclude:</p>
  <xi:include href="inner_not_in_database.xml">
    <xi:fallback><p>Файл не найден</p></xi:fallback>
  </xi:include>
</Root>
```

Результат работы:

```
<Root xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>Следующий фрагмент добавлен с помощью XInclude:</p>
  <p>Файл не найден</p>
</Root>
```

Так как файл не был найден, то выполнен элемент xi:fallback. Однако, если указать корректное имя файла, то элемент xi:fallback не будет выполняться.

**Файл fallback\_2.xml:**

```
<?xml version="1.0" encoding="utf-8"?>
<Root xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>Следующий фрагмент добавлен с помощью XInclude:</p>
  <xi:include href="inner.xml">
    <xi:fallback><p>Файл не найден</p></xi:fallback>
  </xi:include>
```

</Root>

Результат работы:

```
<Root xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>Следующий фрагмент добавлен с помощью XInclude:</p>
  <InnerDocument>
    <p>Это вложенный документ</p>
    <p id="id_inner">Это вложенный документ, id
элемента=id_inner </p>
    <p>Это также вложенный документ</p>
  </InnerDocument>
</Root>
```

### 6.4.3 Добавление результата работы серверного сценария

С помощью элемента `xi:include` можно вставлять не только статические XML-документы, но и документы, которые генерируются динамически с помощью серверного XQuery-сценария.

**Файл `include_query.xql`:**

```
xquery version "1.0";
(: Импорт стандартных модулей :)
declare namespace request="http://exist-db.org/xquery/request";
(: Формат в котором результат должен отображаться в браузере :)
declare option exist:serialize "method=xhtml media-type=text/xml
indent=no";
(: Параметр, передаваемый при вызове из XInclude, должен быть
объявлен как глобальная внешняя переменная :)
declare variable $param external;

(: ++++++ :+)
(: Основной запрос :)
(: ++++++ :+)
<query_result>
  <param>{$param}</param>
</query_result>
```



**Файл include\_query.xml:**

```
<?xml version="1.0" encoding="utf-8"?>
<Root xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>Следующий фрагмент добавлен с помощью XInclude:</p>
  <xi:include href="include_query.xml?param=текст параметра"/>
</Root>
```

**Результат работы:**

```
<Root xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>Следующий фрагмент добавлен с помощью XInclude:</p>
  <query_result>
    <param>текст параметра</param>
  </query_result>
</Root>
```

Элемент `xi:include` был заменен результатом работы серверного сценария. Параметр, переданный сценарию, добавлен в результирующий документ.

**6.4.4 Добавление фрагмента документа с помощью XPointer**

С помощью XPointer можно добавить не весь документ, а его фрагмент.

**Файл xpointer\_1.xml:**

```
<?xml version="1.0" encoding="utf-8"?>
<Root xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>Следующий фрагмент добавлен с помощью XInclude:</p>
  <xi:include href="inner.xml"
xpointer="xpointer(/p[@id='id_inner'])" />
</Root>
```

Для добавления фрагмента документа мы используем элемент `xi:include` с атрибутом `xpointer`. В этом атрибуте указано, что используется схема `xpointer()`, которая возвращает фрагмент документа по XPath-выражению. В круглых скобках необходимо записать XPath-выражение, возвращающее фрагмент документа.

Здесь из документа «inner.xml» выбирается элемент «p» с атрибутом «id='id\_inner'».

Результат работы:

```
<Root xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>Следующий фрагмент добавлен с помощью XInclude:</p>
  <p id="id_inner">Это вложенный документ, id элемента=id_inner
</p>
</Root>
```

В следующем примере из документа «inner.xml» выбираются все элементы «p», у которых не задан атрибут «id».

**Файл xpointer\_2.xml:**

```
<?xml version="1.0" encoding="utf-8"?>
<Root xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>Следующий фрагмент добавлен с помощью XInclude:</p>
  <xi:include href="inner.xml" xpointer="xpointer(//p[not(@id)])"/>
</Root>
```

Результат работы:

```
<Root xmlns:xi="http://www.w3.org/2001/XInclude">
  <p>Следующий фрагмент добавлен с помощью XInclude:</p>
  <p>Это вложенный документ</p>
  <p>Это также вложенный документ</p>
</Root>
```

## 6.5 Материалы для дальнейшего изучения

Рекомендуется ознакомиться с руководством СУБД «eXist» по использованию XInclude, которое содержит дополнительные примеры (руководство находится в разделе документации «eXist»).

Также можно ознакомиться со стандартами XInclude [XInclude, 2006] и стандартами XLink [XLink, 2010] и Xpointer [XPointer Framework, 2003], [XPointer xpointer() Scheme, 2002], [XPointer xmlns() Scheme, 2003], [XPointer element() Scheme, 2003].

## 6.6 Контрольные вопросы

1. Для чего предназначены технологии XInclude, XLink, XPointer?

2. Как используется элемент `xi:include`?
3. Как используется элемент `xi:fallback`?
4. Каким образом с помощью элемента `xi:include` можно добавить результат работы серверного сценария?
5. Как добавить фрагмент документа с использованием `XPointer`?

## 7 Часть 7. Разработка веб-приложений на основе технологии XRХ

В этой главе мы рассмотрим основные принципы построения XRХ-приложений и разработаем пример такого приложения.

Авторы старались разработать эту главу таким образом, чтобы она стала комплексным примером использования рассмотренных ранее XML-технологий.

### 7.1 Архитектура «классического» веб-приложения

Существует большое количество технологий для разработки веб-приложений: ASP.NET, JSP, PHP и многие другие. В рамках каждой из этих технологий существует большое количество библиотек, фреймворков, которые облегчают разработку веб-приложений.

В существующих технологиях в качестве СУБД обычно используется реляционная СУБД. Иногда используются средства объектно-реляционного отображения (ORM – object-relational mapping) для того, чтобы более удобным способом осуществлять преобразования между данными реляционных таблиц и объектами классов объектно-ориентированного языка.

Обобщенная трехзвенная архитектура «классического» веб-приложения приведена на следующем рисунке:

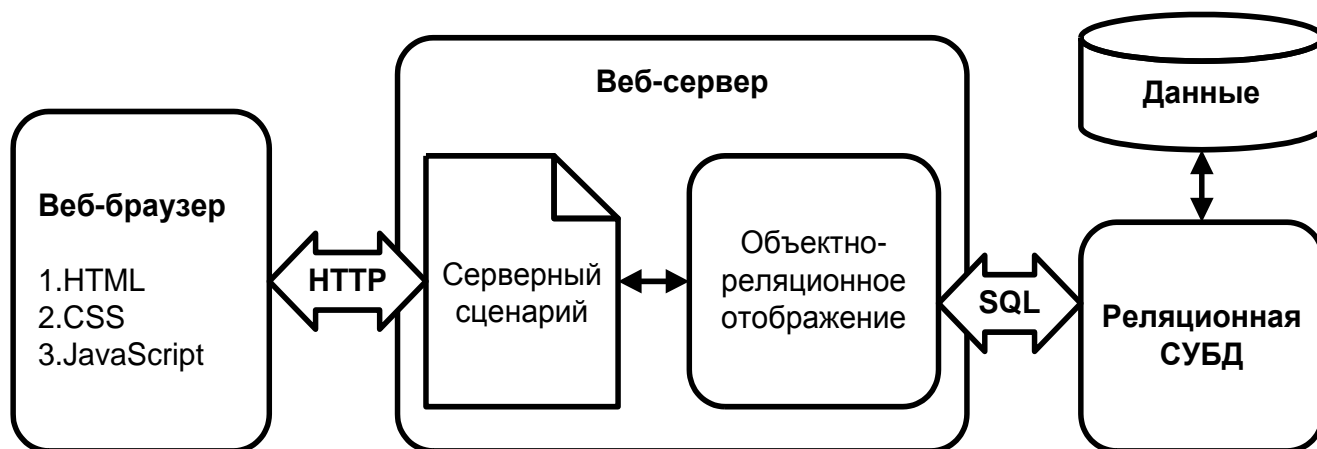


Рис. 7.1. Обобщенная архитектура «классического» веб-приложения.

Первым звеном является веб-браузер, вторым звеном является веб-сервер, третьим звеном является сервер СУБД.

Приведенная схема является очень упрощенной. На ней не показаны возможные дополнительные библиотеки, фреймворки. Также иногда в эту архитектуру добавляется четвертое звено – сервер приложений, который размещается между веб-сервером и сервером СУБД.

Основной вывод, который хотелось бы сделать из этой схемы, состоит в том, что для разработки веб-приложения необходимо одновременно использовать много различных технологий, что увеличивает сложность разработки.

На стороне веб-браузера могут использоваться: HTML (1), каскадные таблицы стилей CSS (2), JavaScript (3). В случае использования AJAX-технологии на JavaScript разрабатывается значительная часть веб-приложения, а серверные сценарии в основном только формируют данные.

На стороне веб-сервера могут использоваться: какая-либо из технологий «серверных страниц» (например ASP.NET, JSP) (4), процедурный или объектно-ориентированный язык программирования (например C# или Java) (5). Обращение к СУБД производится с использованием языка SQL (6) или с помощью средств объектно-реляционного отображения (7).

На стороне СУБД может использоваться какой-либо язык для написания хранимых процедур (8).

Если мы хотим работать с данными в XML-формате (9), то это приводит к необходимости написания дополнительного программного кода. При этом часто используется технология XSLT (10).

Итого 10 технологий, хотя на практике их может быть намного больше.

Таким образом, при разработке «классических» веб-приложений приходится использовать довольно много различных технологий, которые необходимо соединить друг с другом, что также требует дополнительных усилий.

Еще одна трудность состоит в том, что на каждом «звене» используется собственная модель данных (СУБД – реляционная, веб-сервер – объектно-ориентированная, веб-браузер – HTML). Приходится тратить дополнительные

усилия для преобразования данных между различными моделями, при этом могут дополнительно возникать ошибки преобразования данных.

Мы ни в коем случае не критикуем «классическую» архитектуру. Она сложилась исторически, и в большинстве случаев ее приходится использовать.

Но если мы хотим использовать только XML-данные, то, может быть, возможно упростить эту архитектуру?

Технология XRХ и является упрощением «классической» архитектуры для случая использования XML. Если говорить точнее, то архитектура XRХ является «надстройкой» над «классической» архитектурой, которая позволяет более простым способом разрабатывать веб-приложения на основе XML.

## **7.2 Архитектура веб-приложения на основе технологии XRХ**

### **7.2.1 Обобщенная архитектура**

Обобщенная архитектура веб-приложения на основе технологии XRХ приведена на следующем рисунке:

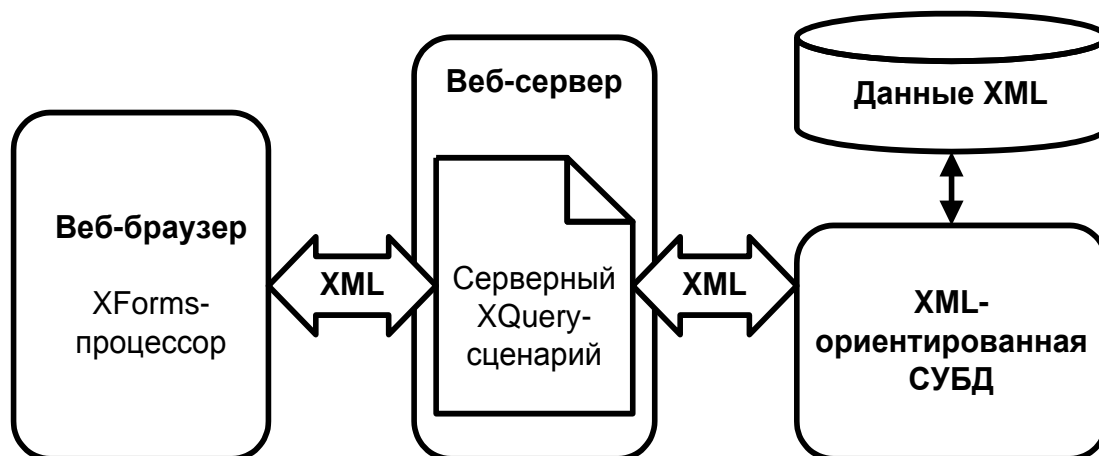


Рис. 7.2. Обобщенная архитектура веб-приложения на основе технологии XRХ.

Аббревиатура XRХ расшифровывается как XForms-REST-XQuery.

- XForms – используется на стороне веб-браузера для ввода и редактирования форм.

Фактически XForms-процессор – это «толстый» клиент для ввода форм, который реализован с использованием веб-технологий.

Как и в случае HTML-форм, формы на XForms могут быть статическими, а могут динамически генерироваться на веб-сервере с помощью XQuery-сценариев.

- REST – означает, что XML-данные между формами на XForms и XQuery-сценариями передаются с помощью HTTP-протокола, чаще всего используются «обычные» методы GET и POST.

Более сложной альтернативой REST являются технологии «классических» веб-сервисов (SOAP-протокол и другие технологии). В XRX-технологии они, как правило, не используются.

Термин REST (Representational State Transfer) был введен Р.Филдингом (одним из разработчиков протокола HTTP) для обозначения одной из возможных моделей сетевых протоколов [Fielding, 2000]. Протокол HTTP построен на модели REST.

В аббревиатуре XRX термин «REST» фактически означает использование протокола HTTP.

- XQuery – используется на стороне веб-сервера для доступа к данным и в качестве языка программирования серверных сценариев.

Если в «классической» архитектуре на стороне веб-сервера используется два языка, один для программирования серверных сценариев (например C#), другой для доступа к данным (SQL), то в XRX-архитектуре обе функции выполняет XQuery.

Также с помощью XQuery динамически генерируются XForms-формы.

Для разработки XRX-приложений необходимо использовать следующие технологии:

1. XML
2. XQuery
3. XForms
4. HTML / XHTML

Также дополнительно могут быть использованы:

5. XSLT
6. DTD / XML-схемы

## 7. CSS

## 8. JavaScript

Количество технологий почти такое же, как в случае «классических» веб-приложений. Однако большая часть этих технологий ориентирована на использование модели данных XML.

В чем основное преимущество технологии XRX при использовании XML-данных? В «классической» архитектуре на каждом «звене» использовалась собственная модель данных. В XRX-технологии на всех «звеньях» используется XML. То есть не требуется преобразовывать данные между различными моделями данных, преобразовывать нужно только данные в модели XML.

Рассмотрим взаимодействие между тремя «звеньями» более подробно:

1. На стороне клиента XForms-процессор принимает от XQuery-сценария описание формы в формате XML (форма на XForms, как правило, является XHTML-документом). После заполнения формы пользователем введенные данные представляют собой XML-фрагмент, и этот фрагмент передается на веб-сервер. XML-фрагмент может быть непосредственно сохранен в XML-ориентированную СУБД или может быть передан серверному XQuery-сценарию для дальнейшей обработки.

2. На стороне веб-сервера также используются данные в формате XML, которые обрабатываются с помощью XQuery-сценариев. XQuery-сценарий может взаимодействовать с XML-ориентированной СУБД, читать и записывать данные в формате XML.

Серверный XQuery-сценарий при взаимодействии с клиентской частью может генерировать, например, следующие виды документов:

- Формы на XForms, которые передаются для ввода данных XForms-процессору.
- XML-документы произвольного вида. Например, это могут быть XHTML-документы, которые передаются в браузер и отображаются пользователю. XML-документы могут дополнительно обрабатываться с помощью XSLT-преобразований (в eXist это встроенная функция).

[Оглавление](#)



3. XML-ориентированная СУБД хранит данные в формате XML. Возможна выборка, добавление, обновление, удаление XML-данных.

Таким образом, при переходе между звеньями не требуется преобразовывать данные между различными моделями, так как везде используется XML.

### **7.2.2 Проверка введенных данных**

В «классическом» веб-приложении пользователь заполняет поля в HTML-форме, и они передаются на сервер в виде набора переменных (имя=значение).

Обычно основная проверка правильности введенных данных производится в серверном сценарии на веб-сервере. Иногда проверка дополнительно производится в браузере с помощью JavaScript, но поскольку поддержка JavaScript в браузере может быть отключена, то основную проверку введенных данных всегда рекомендуется делать в серверном сценарии.

В случае XRX-приложения проверка правильности введенных данных обычно производится на клиентской стороне при заполнении XForms-формы. При этом используются средства проверки данных, которые предусмотрены в технологии XForms. Затем введенные данные передаются на сервер в виде XML-фрагмента.

Далее на сервере можно провести дополнительную проверку данных в XML-фрагменте.

Дополнительная серверная проверка данных может быть полезна в следующих случаях:

1. Необходимо проверить сложные зависимости между данными, которые трудно описать с помощью `xforms:bind` (`xforms:bind` обычно описывает ограничения для одного XML-элемента).

2. При проверке необходимо использовать данные из БД.

Серверная проверка используется только в крайнем случае, так как в результате серверной проверки необходимо сформировать сообщение об ошибке, заново сгенерировать XForms-форму, что достаточно трудоемко. Поэтому по возможности стараются использовать средства проверки данных, встроенные в

XForms, и только в крайнем случае используется дополнительная серверная проверка.

### 7.2.3 Детализированная архитектура

Детализированная архитектура веб-приложения на основе технологии XRX приведена на следующем рисунке:

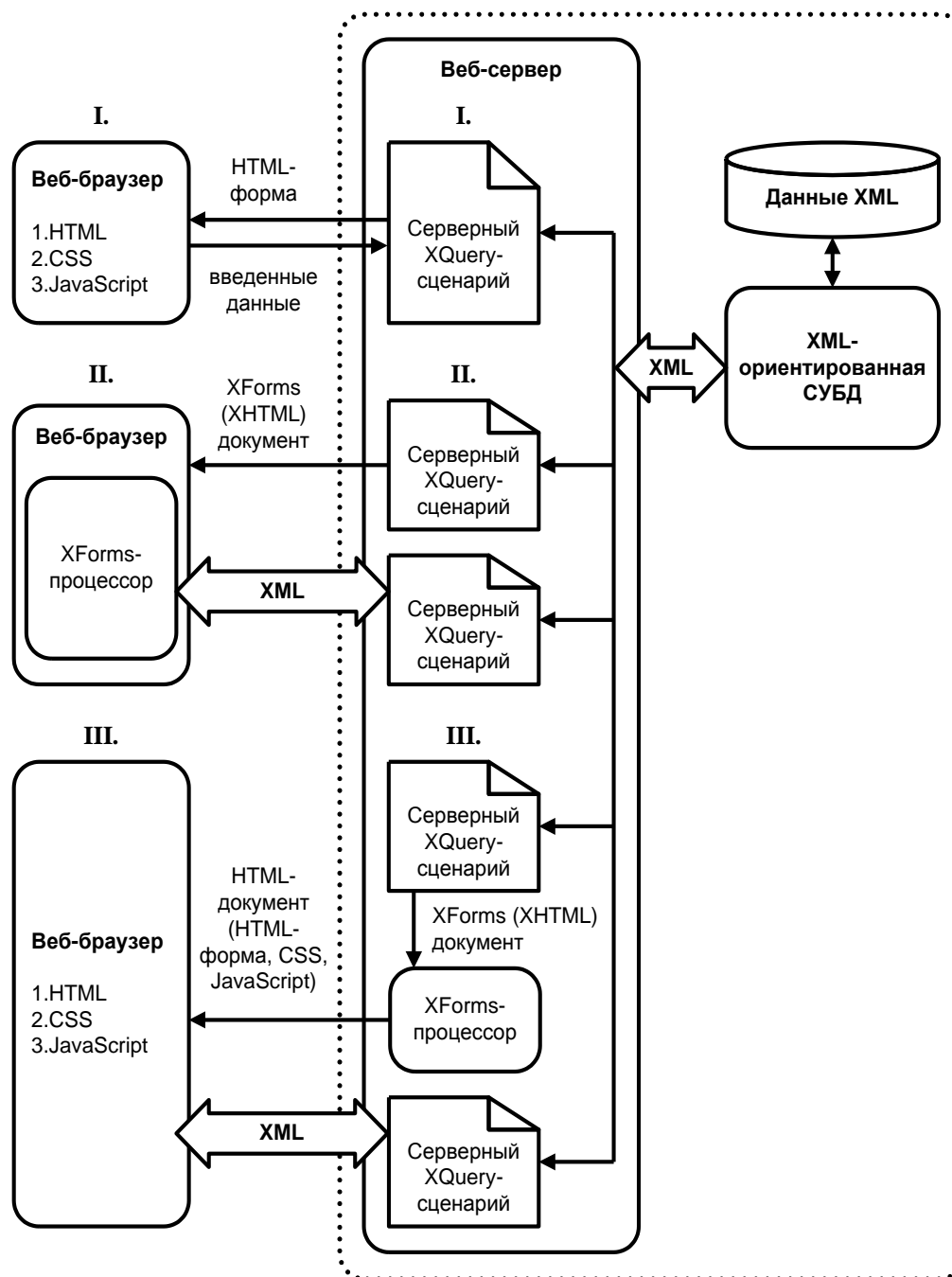


Рис. 7.3. Детализированная архитектура веб-приложения на основе технологии XRX.

На рисунке приведены три наиболее традиционных варианта архитектуры XRX-приложения. Можно также предложить множество других вариантов архитектуры.

Вариант I соответствует упрощенной XRX-технологии (без использования XForms).

Серверный XQuery-сценарий генерирует обычный HTML (или XHTML) документ, который содержит обычную HTML-форму. Введенные в форму данные в виде набора переменных (имя=значение) передаются на сервер и обрабатываются XQuery-сценарием. Технология XForms при этом не используется.

Вариант II соответствует случаю, когда XForms-процессор реализован в виде клиентской технологии, например, встроен в браузер (или является плагином к браузеру).

Этому случаю соответствует XForms-процессор XSLTForms (встроенный в eXist), который реализован в виде XSLT-преобразования, и предположительно это преобразование делается в браузере. (Если XSLT-преобразование делать на стороне веб-сервера в XQuery-сценарии, то это ближе к третьему варианту архитектуры.)

В случае варианта II серверный XQuery-сценарий генерирует XForms-форму (которая, как правило, является частью XHTML-документа) и отправляет эту форму XForms-процессору.

Эта форма может быть статическим документом, однако в варианте II наиболее привлекательной является именно возможность динамического формирования XForms-документов.

В начале работы XForms-форма может читать необходимые данные по умолчанию, а после заполнения отправляет на сервер введенные данные в формате XML.

Часто для генерации XForms-формы используется один сценарий, а для обмена данными с формой используется другой сценарий (в нашем примере приложения мы предполагаем делать именно так).

Однако можно также использовать один сценарий, который выполняет все функции, или три сценария (генерация формы, чтение данных, запись данных).

Здесь можно отметить отличие XForms-форм от классических HTML-форм. В HTML-форме нам потребовалось бы выполнить в серверном сценарии только два действия:

1. Генерация HTML формы вместе с данными по умолчанию, так как данные по умолчанию сохраняются прямо в тэгах HTML.

2. Обработка данных на сервере после заполнения формы, данные передаются на сервер в виде набора переменных (имя=значение).

В случае XForms требуется выполнить три действия:

1. Генерация XForms-формы без данных по умолчанию. (Возможно также сгенерировать XForms-форму с данными по умолчанию, но при работе с БД такой вариант не очень удобен.)

2. После того как форма будет загружена, модель данных XForms-формы запросит с сервера данные по-умолчанию. Необходимо разработать серверный сценарий, который вернет данные по-умолчанию для формы.

3. Обработка данных на сервере после заполнения формы, данные передаются на сервер в виде XML-фрагмента.

Вариант III соответствует случаю, когда XForms-процессор реализован в виде отдельного серверного веб-приложения. Примерами таких процессоров являются Orbeon Forms и betterFORM.

В этом случае серверный XQuery-сценарий генерирует XForms-форму и каким-либо образом вызывает XForms-процессор как библиотеку. (В нашем примере мы предполагаем таким образом использовать XSLTForms, вызывая в серверном сценарии XSLT-преобразование).

В варианте III могут возникнуть трудности с динамической генерацией XForms-форм, так как некоторые процессоры могут открывать только статические XHTML-документы.

XForms-процессор на основе переданной ему XForms-формы генерирует HTML-документ, содержащий HTML-форму, стили CSS, сценарии JavaScript и

передает HTML-документ в браузер. (В XSLTForms для выполнения этого действия необходимо обработать XHTML-документ, содержащий XForms-форму, с помощью XSLT-преобразования.)

Далее сгенерированный HTML-документ работает с данными так же, как в варианте II. Как правило, для взаимодействия с сервером используется AJAX-технология.

Основная разница между вариантами II и III заключается в том, что в случае варианта II обработка формы (и, возможно, ее преобразование в HTML, CSS, JavaScript) производится на стороне веб-браузера, а в случае варианта III на стороне веб-сервера.

Хочется также отметить, что в XML-ориентированных СУБД (например, в eXist) веб-сервер может быть совмещен с сервером СУБД, что показано на рисунке пунктиром.

eXist устанавливается со встроенным веб-сервером Jetty, или может быть установлен на любой веб-сервер с поддержкой Java в виде веб-архива. При этом функции выполнения XQuery-сценариев и прямого доступа к данным в eXist не отделены друг от друга.

Фактически eXist представляет собой сервер приложений с возможностью разработки веб-приложений на XQuery и с возможностью хранения данных в формате XML.

#### **7.2.4 Ограничения XRX-технологии**

XRX не является универсальной технологией, которая могла бы полностью заменить «классическую» технологию разработки веб-приложений.

К основным ограничениям технологии XRX можно отнести следующие:

1. Эффективна только при работе с данными XML и XML-ориентированной БД.
2. Основной проблемой существующих open-source XML-ориентированных БД является их надежность и возможность хранения больших объемов данных.

Пока по этим параметрам их трудно сравнивать с такими СУБД, как Microsoft SQL Server или Oracle.

Однако эту проблему можно преодолеть, если использовать коммерческие XML-ориентированные СУБД (например, Tamino).

Также можно для хранения данных использовать Microsoft SQL Server или Oracle. Эти СУБД позволяют хранить данные в формате XML (есть встроенный тип данных), а также в них есть встроенная поддержка XQuery. Надежность хранения в этом случае повышается, однако необходимо доработать часть приложения, которая отвечает за REST-доступ.

3. Технология XForms не всегда позволяет достаточно гибко реализовать проверку данных в форме. Иногда для проверки данных в XForms требуется разрабатывать достаточно сложные XPath-выражения. Иногда требуется дополнительная проверка данных в серверном сценарии.

4. Для технологии XRX пока отсутствуют интегрированные среды разработки. Для разработки XQuery-сценариев и XForms-документов можно использовать коммерческие XML-редакторы (Altova XMLSPY, Oxygen XML). В eXist для этих целей может использоваться среда разработки eXide.

## **7.3 Пример разработки веб-приложения на основе технологии XRX**

### **7.3.1 Постановка задачи**

Необходимо разработать веб-приложение, которое позволяет вводить данные о компьютерах, а также формировать отчеты на основе введенных данных.

### **7.3.2 Логическая модель данных**

#### **7.3.2.1 Описание сущностей**

Логическая модель данных содержит три сущности.

1. Сущность «Тип процессора».

Атрибуты:

1.1. Код процессора (уникальный идентификатор).

1.2. Наименование процессора (текст).

2. Сущность «Компьютер».

Атрибуты:

2.1. Код компьютера (уникальный идентификатор).

2.2. Код процессора (связь с сущностью «Тип процессора»).

2.3. Наименование компьютера (текст).

2.4. Объем ОП в Мб (целое число).

2.5. Объем жесткого диска в Гб (целое число).

2.6. Используется как сервер (логический тип).

3. Сущность «Модернизация компьютера».

Атрибуты:

3.1. Код модернизации (уникальный идентификатор).

3.2. Код компьютера (связь с сущностью «Компьютер»).

3.3. Дата модернизации (дата).

3.4. Описание модернизации (текст).

«Модернизация компьютера» является «слабой» сущностью, не может существовать без сущности «Компьютер».

#### **7.3.2.2 Описание связей**

Между сущностями «Тип процессора» и «Компьютер» существует связь «Компьютер содержит процессор» типа «один-ко-многим», один тип процессора может использоваться в нескольких компьютерах.

Между сущностями «Компьютер» и «Модернизация компьютера» существует связь «Компьютер модернизируется» типа «один-ко-многим», один компьютер может модернизироваться много раз.

#### **7.3.2.3 Диаграмма сущность-связь**

Диаграмма сущность-связь для рассмотренной предметной области приведена на следующем рисунке:

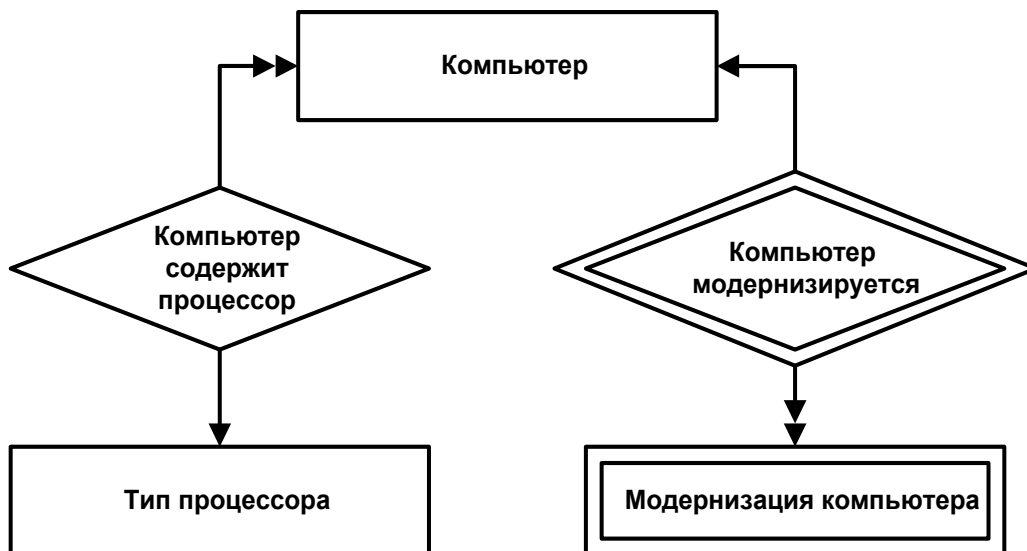


Рис. 7.4. Диаграмма сущность-связь.

### 7.3.3 Физическая модель данных в СУБД «eXist»

В СУБД «eXist» сущность может быть представлена тремя способами:

Способ 1. Сущности соответствует XML-файл.

Способ 2. Сущности соответствует коллекция XML-файлов.

Способ 3. Зависимая сущность может храниться в одном файле с основной сущностью.

Можно также придумать другие способы хранения. Например, в одном файле могут храниться несколько сущностей (или вообще все сущности), но такой способ не очень удобен (например, неудобно выгружать фрагменты данных из БД с помощью WebDAV).

Способ 1. Одной сущности (в реляционной модели это таблица) соответствует один файл XML, расположенный в определенной коллекции.

Экземпляру сущности (в реляционной модели это запись таблицы) соответствует XML-элемент, который сохранен в файле XML.

Атрибуту сущности (в реляционной модели это поле записи) соответствует XML-элемент, который вложен в XML-элемент, соответствующий экземпляру.

Название файла обычно совпадает с названием сущности.



Способ 1 удобен для хранения небольших сущностей, структура которых не должна часто изменяться, например, справочников.

### Пример 7.1. Описание сущности в виде документа XML:

```
<?xml version = "1.0" encoding = "utf-8"?>
<Сущность>
  <Экземпляр_сущности id="Экземпляр_1">
    <Атрибут1>Значение 1</Атрибут1>
    ...
    <АтрибутK>Значение K</АтрибутK>
  </Экземпляр_сущности>
  ...
  <Экземпляр_сущности id="Экземпляр_N">
    <Атрибут1>Значение N 1</Атрибут1>
    ...
    <АтрибутK>Значение N K</АтрибутK>
  </Экземпляр_сущности>
</Сущность>
```

Способ 2. Сущности соответствует коллекция XML-ориентированной СУБД.

Экземпляру сущности соответствует один файл XML, расположенный в коллекции, соответствующей сущности.

В качестве названия файла обычно используется значение атрибута id с расширением xml.

Атрибуту сущности соответствует XML-элемент, который сохранен в файле XML.

### Пример 7.2. Описание экземпляра сущности в виде документа XML:

Файл «Экземпляр\_1.xml»:

```
<?xml version="1.0" encoding="utf-8"?>
<Экземпляр_сущности id="Экземпляр_1">
  <Атрибут1>Значение 1</Атрибут1>
  ...
  <АтрибутK>Значение K</АтрибутK>
</Экземпляр_сущности>
```

Способ 3. Экземпляры зависимой (слабой) сущности можно встраивать в экземпляры основной сущности в виде вложенных элементов XML.

Экземпляры основной сущности могут храниться с помощью способов 1 или 2 (в следующем примере для хранения основной сущности используется способ 2).

### Пример 7.3. Описание слабой (зависимой) сущности:

```
<?xml version="1.0" encoding="utf-8"?>
<Экземпляр_сущности id="Экземпляр_1">
  <Атрибут1>Значение 1</Атрибут1>
  ...
  <АтрибутK>Значение K</АтрибутK>
  <Экземпляры_зависимой_сущности>
    <Экземпляр_зависимой_сущности
id="Экземпляр_зависимой_сущности_1">
      <Атрибут_зависимой_сущности1>Значение зависимой сущности
1</Атрибут_зависимой_сущности1>
      ...
      <Атрибут_зависимой_сущностиK>Значение зависимой сущности
K</Атрибут_зависимой_сущностиK>
    </Экземпляр_зависимой_сущности>
    ...
  </Экземпляры_зависимой_сущности>
</Экземпляр_сущности>
```

### 7.3.4 Отображение логической модели данных в физическую модель данных

1. Данные для нашего приложения будут храниться в коллекции «db/иу5/xrx».

2. Во вложенных коллекциях будут храниться данные о процессорах («db/иу5/xrx/processors») и компьютерах («db/иу5/xrx/computers»).

3. Сущность «Тип процессора» будем хранить в БД с помощью способа 1.

Для хранения сущности будем использовать файл «processors.xml» в коллекции «/db/iu5/xrx/processors».

Описание XML-файла приведено в следующей таблице:

Таблица 7.1. Описание XML-файла для сущности «Тип процессора».

Инфологическая модель	Датологическая модель
Сущность «Тип процессора»	Файл «processors.xml». Корневой XML-элемент файла «processors»
Экземпляр сущности	XML-элемент «processors/processor»
Атрибут сущности «Код процессора»	XML-атрибут «processors/processor/@id»
Атрибут сущности «Наименование процессора»	XML-элемент «processors/processor/ processor_name»

В следующем примере представлен справочник, содержащий данные:

#### Пример 7.4. Пример файла «processors.xml»

```
<?xml version="1.0" encoding="UTF-8"?>
<processors>
  <processor id="333">
    <processor_name>Процессор 1</processor_name>
  </processor>
  <processor id="334">
    <processor_name>Процессор 2</processor_name>
  </processor>
  ...
</processors>
```

4. Сущность «Компьютер» будем хранить с помощью способа 2, а сущность «Модернизация компьютера» с помощью способа 3.

Для хранения будем использовать коллекцию «/db/iu5/xrx/ computers».

Каждому XML-файлу в этой коллекции будут соответствовать данные об одном компьютере. Название файла имеет структуру «id компьютера.xml».

Данные о модернизации компьютера будут также храниться в XML-файле, соответствующем компьютеру.

Таблица 7.2. Описание XML-файла для сущностей «Компьютер» и «Модернизация компьютера».

<b>Инфологическая модель</b>	<b>Датологическая модель</b>
Сущность «Компьютер»	Коллекция «/db/iu5/xrx/ computers».
Экземпляр сущности	Файл «id компьютера.xml». Корневой XML-элемент файла «computer»
Атрибут сущности «Код компьютера»	XML-атрибут «computer/@id»
Атрибут сущности «Код процессора»	XML-элемент «computer/processor_id»
Атрибут сущности «Наименование компьютера»	XML-элемент «computer/computer_name»
Атрибут сущности «Объем ОП в Мб»	XML-элемент «computer/op»
Атрибут сущности «Объем жесткого диска в Гб»	XML-элемент «computer/hdd»
Атрибут сущности «Используется как сервер»	XML-элемент «computer/server»
Сущность «Модернизация компьютера»	XML-элемент «computer/upgrades»
Экземпляр сущности	XML-элемент «computer/upgrades/upgrade»
Атрибут сущности «Код модернизации»	XML-атрибут «computer/upgrades/upgrade/@id»
Атрибут сущности «Дата модернизации»	XML-элемент «computer/upgrades/upgrade/

Инфологическая модель	Датологическая модель
	upgrade_date»
Атрибут сущности «Описание модернизации»	XML-элемент «computer/upgrades/upgrade/ upgrade_description»

### Пример 7.5. Пример файла, содержащего данные о компьютере

```
<?xml version="1.0" encoding="utf-8"?>
<computer id="3e915ce2-a97a-465c-8ad9-
f7d8f5bb44b9_2010_10_18_0_56_10F843">
  <computer_name>Сервер 1</computer_name>
  <processor_id>29435348-3aa3-4cb5-9e3e-
84d4b26e83cb_2010_10_18_0_55_46F809</processor_id>
  <op>1024</op>
  <hdd>1000</hdd>
  <server>true</server>
  <upgrades>
    <upgrade/>
  </upgrades>
</computer>
```

### Пример 7.6. Пример файла, содержащего данные о компьютере и его модернизациях

```
<?xml version="1.0" encoding="utf-8"?>
<computer id="3e915ce2-a97a-465c-8ad9-
f7d8f5bb44b9_2010_10_18_0_56_10F843">
  <computer_name>Сервер 1</computer_name>
  <processor_id>29435348-3aa3-4cb5-9e3e-
84d4b26e83cb_2010_10_18_0_55_46F809</processor_id>
  <op>1024</op>
  <hdd>1000</hdd>
  <server>true</server>
  <upgrades>
```

```

<upgrade/>
<upgrade id="1_2010-10-18T22:46:37Z">
  <upgrade_date>2010-10-04</upgrade_date>
  <upgrade_description>увеличение ОП</upgrade_description>
</upgrade>
<upgrade id="1_2010-10-18T22:46:54Z">
  <upgrade_date>2010-10-05</upgrade_date>
  <upgrade_description>замена диска</upgrade_description>
</upgrade>
</upgrades>
</computer>

```

5. Связи между сущностями могут быть заданы двумя способами – в виде равенства соответствующих XML-элементов (XML-атрибутов) или в виде вложенности элементов.

Связь «Компьютер содержит процессор» между сущностями «Тип процессора» и «Компьютер» задается в виде равенства XML-атрибута «processors/processor/@id» (в XML-файле, соответствующем процессору) и XML-элемента «computer/processor\_id» (в XML-файле, соответствующем компьютеру). Таким образом, связь задается в виде равенства элементов (атрибутов), которые находятся в разных XML-файлах (что похоже на реляционную модель, где должны совпадать поля различных таблиц).

Связь «Компьютер модернизируется» между сущностями «Компьютер» и «Модернизация компьютера» задается в виде вложенности элементов «upgrade» (соответствующих модернизации компьютера) в элемент «computer» (соответствующий компьютеру). Такой способ может быть использован в случае слабой (зависимой) сущности (данные зависимой сущности вложены в данные основной сущности). Этот способ не имеет прямых аналогий в реляционной модели.

### 7.3.5 Разработка приложения

**Пример 7.7** содержит каталог «xrx», содержащий файлы XRX-приложения.

Для работы примера необходимо создать коллекцию «/db/iu5/xrx/» и загрузить в коллекцию файлы примера. При этом необходимо создать подколлекции «styles» и «xsltforms» и загрузить в них соответствующие файлы примера.

Коллекция «styles» содержит CSS-стили, а коллекция «xsltforms» версию процессора XSLTForms, которая используется в данном примере. Из-за изменений в версиях XSLTForms использование других версий может привести к некорректной работе примера.

Пример вызывается с использованием URL:  
<http://localhost:8080/exist/rest/db/iu5/xrx/index.xql>

#### 7.3.5.1 Соглашения по наименованию элементов XRX-приложения

В XRX-приложениях обычно не используют сложную систему настроек.

Принято использовать определенные договоренности по наименованию файлов, сценариев и использовать эти имена непосредственно в тексте сценариев.

Конечно, такая система «хардкодинга» является менее гибкой по сравнению с технологией конфигурационных файлов (которая используется, например, в ASP.NET), но для разработки не очень сложных веб-приложений это достаточно простой и надежный способ.

В нашем примере будем использовать следующую систему договоренностей.

В веб-приложении создается один XQuery-модуль (файл с расширением .xqm), в котором объявляются константы, пути к файлам, необходимые библиотечные функции и т.д. В нашем примере файл называется «module.xqm». Этот модуль используется в каждом XQuery-сценарии.

Для каждой сущности (в нашем примере для сущностей «Тип процессора» и «Компьютер») создается три XQuery-сценария:

1. list\_\*.xql – сценарий возвращает список (таблицу) объектов.

[Оглавление](#)

2. edit\_\*.xql – редактирование одного объекта.

3. data\_\*.xql – работа с данными (сохранение, удаление и т.д.).

Здесь «\*» - название сущности. Такую совокупность сценариев будем называть предметным «модулем» (только это не XQuery-модуль, а «модуль» сценариев, соответствующих одной сущности предметной области). Все сценарии предметного модуля содержат в имени файла название сущности (модуля).

Дальше мы несколько расширим эту схему, и «\*» будет включать не только название, но и версию модуля.

### 7.3.5.2 Использование шаблона проектирования «модель-вид-контроллер»

Шаблон проектирования «модель-вид-контроллер» (Model-View-Controller, MVC) позволяет при проектировании системы различать:

- Данные приложения (модель).
- Пользовательский интерфейс (вид).
- Управляющую логику (контроллер).

Часто используются вариации этого шаблона проектирования, адаптированные для конкретной технологии или приложения [MVC, 2007].

Мы будем использовать разновидность этого шаблона при разработке XRX-приложения.

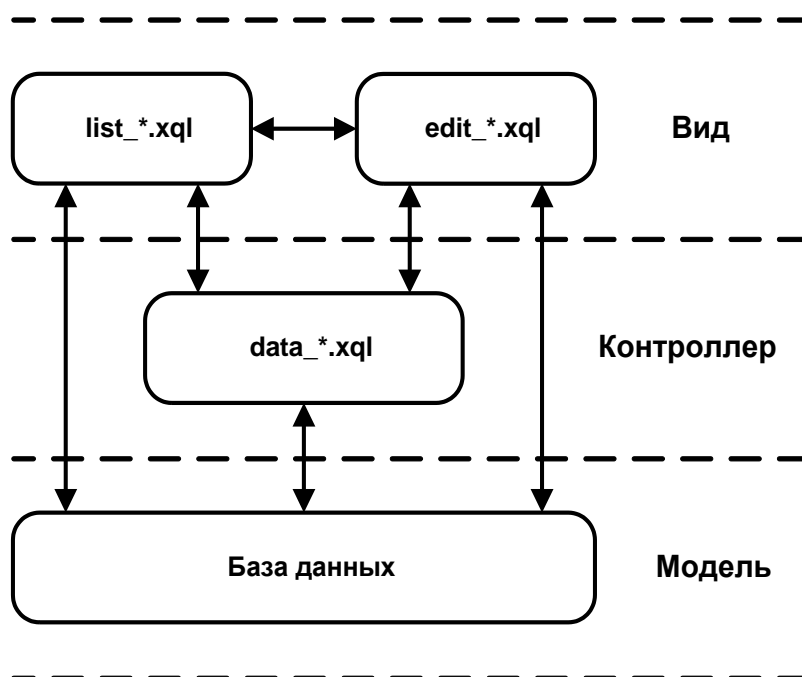




Рис. 7.5. Использование шаблона проектирования «модель-вид-контроллер».

Наше приложение не является «строгой» реализацией этого шаблона, мы использовали только основные принципы шаблона «модель-вид-контроллер».

Функцию модели выполняют XML-данные, хранящиеся в БД.

Функцию контроллера выполняет сценарий «data\_\*.xql», реализующий работу с данными (добавление, обновление, удаление, формирование вспомогательных данных и т.д.).

Используется два «вида» (что допускается шаблоном MVC). Сценарий «list\_\*.xql» возвращает список (таблицу) объектов, а сценарий «edit\_\*.xql» осуществляет редактирование одного объекта.

«Виды» могут взаимодействовать друг с другом, с контроллером, с моделью.

#### **7.3.5.3 Использование редактора XQuery-сценариев**

В качестве редактора XQuery-сценариев используется eXide.

Для «eXist» важно, чтобы в XQuery-сценариях в качестве кодировки символов использовался вариант «UTF-8 без BOM». XQuery-сценарии, сохраненные в другой кодировке, могут вызывать ошибки.

Более подробно с особенностями кодировки UTF-8 и использованием BOM (Byte Order Mark) можно ознакомиться на сайте стандарта Unicode (<http://www.unicode.org>).

#### **7.3.5.4 Справочник «Тип процессора»**

Так как справочник содержит только одно поле ввода (наименование процессора), то будем использовать вариант архитектуры I, то есть технология XForms использоваться не будет, данные будут вводиться с помощью обычных форм HTML.

Для реализации справочника необходимо разработать три сценария:

- list\_processor\_1.2.xql – список типов процессоров (вид).
- edit\_processor\_1.2.xql – редактирование типа процессора (вид).

- data\_processor\_1.2.xql – редактирование типа процессора (контроллер).

Имена файлов сценариев имеют следующую структуру: «тип\_название модуля\_версия модуля.xql».

Тип может содержать значения «list», «edit» или «data». Название модуля – «processor» Версия модуля – «1.2».

В нашем примере мы предусмотрим возможность хранения сценариев модуля в нескольких версиях в одном каталоге, поэтому версия модуля входит в наименование файлов. То есть, если в дальнейшем мы разработаем версию «1.3», то удалять файлы версии «1.2» нет необходимости, они могут храниться в том же каталоге. Если в дальнейшем версия «1.3» будет признана неудачной, то возврат к версии «1.2» будет простым. Эта простейшая «система контроля версий» полезна, если мы не используем какую-либо полноценную систему версионного контроля, например SVN.

#### 7.3.5.4.1 Сценарий list\_processor\_1.2.xql

Сценарий предназначен для вывода списка процессоров и кнопок добавления, редактирования, удаления.

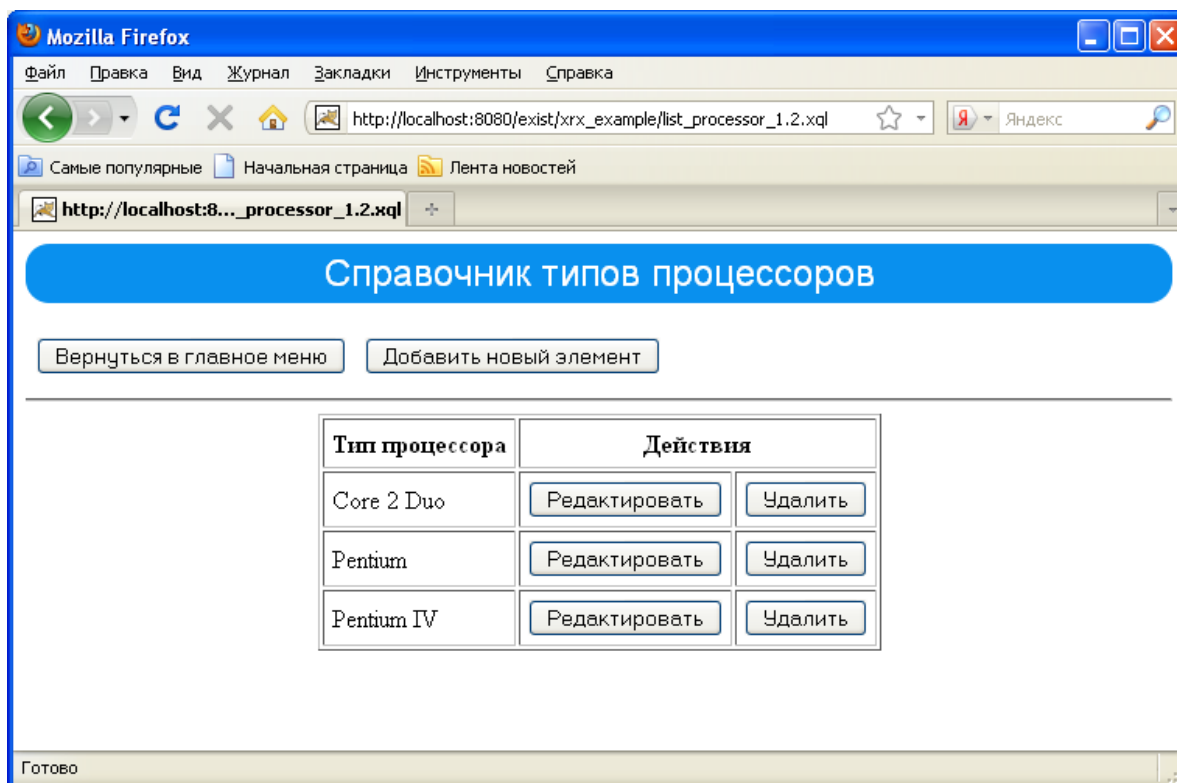


Рис. 7.6. Сценарий list\_processor\_1.2.xql

Сценарий содержит три стандартные части:

1. Пролог сценария.

2. Локальные функции (вспомогательные функции, которые вызываются из основного запроса).

3. Основной запрос.

Пролог сценария:

```
xquery version "1.0";
```

```
(: Импорт стандартных модулей eXist :)
declare namespace request="http://exist-db.org/xquery/request";
declare namespace session="http://exist-db.org/xquery/session";
```

```
(: Импорт модуля приложения :)
import module namespace xrx_example =
"http://iu5.bmstu.ru/edu/xrx_example" at "module.xqm";
```

Здесь задается префикс пространства имен «xrx» (который будет использоваться для обращения к переменным и функциям модуля), идентификатор пространства имен «http://iu5.bmstu.ru/edu/xrx» (который далее в сценарии не используется, он просто «обозначает» пространство имен модуля) и название файла модуля «module.xqm».

```
(: Формат в котором результат должен отображаться в браузере :)
declare option exist:serialize "method=xhtml media-type=text/html
indent=no";
```

Здесь указано, что результат, сгенерированный сценарием, должен быть в формате html.

```
(: Идентификатор модуля :)
declare variable $module_id {"processor"};
(: Версия модуля :)
declare variable $module_ver {"1.2"};
```

Глобальные переменные сценария, которые определяют название модуля (\$module\_id) и версию модуля (\$module\_ver). Эти переменные далее используются как параметры при вызове различных функций (прежде всего это функции, которые формируют гиперссылки на другие сценарии модуля «edit\_processor\_1.2.xql» и «data\_processor\_1.2.xql»).

Для того чтобы наш «версионный контроль» работал корректно, необходимо чтобы переменные \$module\_id и \$module\_ver были правильно установлены во всех сценариях модуля и соответствовали имени файла.

### Основной запрос сценария:

```
(: ++++++ :)  
(: Основной запрос :)  
(: ++++++ :)  
<html>  
  <head>  
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-  
8"/>  
    <style type="text/css">  
      <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"  
href="{xrx_example:PathURI_current_css()}" />  
    </style>  
  </head>  
  <body>  
    <h1>Справочник типов процессоров</h1>  
    { local:main() }  
  </body>  
</html>  
(: ++++++ :)
```

Основной запрос сценария формирует структуру выходного HTML-документа.

В элементе `<style>` мы вставляем таблицу стилей CSS с использованием механизма XInclude. Функция `xrx_example:PathURI_current_css()` (которая определена в библиотечном модуле) возвращает путь к таблице стилей.

Таблица стилей используется не совсем обычная, так как она вставляется с помощью XInclude, то она должна быть правильным XML-документом. Более подробно содержимое таблиц стилей рассмотрено в следующих разделах.

В элементе `<body>` вызывается локальная функция `local:main()`, которая выполняет основные действия.

Локальные функции (они располагаются между прологом и основным запросом):

```
(: ++++++ :)  
(: Основная функция :)  
(: ++++++ :)  
declare function local:main() as node()  
{  
<div>  
  {  
    (local:ShowHeader()),  
    (local:ShowList()),  
    (local:ShowFooter())  
  }  
</div>  
};
```

Функция объявляется с помощью ключевых слов «declare function». Перед названием функции «main» через двоеточие указывается префикс пространства имен. Префикс «local» означает, что это локальная функция.

Входных параметров у функции нет (после названия функции стоят пустые скобки). После скобок указывается тип возвращаемого значения «as тип». Тип «node()» означает, что функция возвращает один элемент XML.

Если бы функция возвращала строку, то было бы указано «as xs:string».

Для указания множественности (по-английски «кардинальности» – cardinality) параметров и возвращаемого значения используются специальные символы (которые используются в DTD) – «\*», «+» и «?».

Если необходимо, чтобы функция возвращала множество элементов (0 или более), то после типа ставится «\*», например «node(\*)» или «as xs:string\*».

Также можно использовать «?» (0 или 1) и «+» (1 или более), например «node(+)» или «node(?)».

Кардинальность параметров может показаться несколько непривычной с точки зрения традиционных языков программирования. Наверное, наиболее близким аналогом кардинальности в традиционных языках можно считать массивы.

Рассмотрим тело функции.

Тело функции ограничено фигурными скобками, после закрывающей скобки ставится точка с запятой.

В нашем случае функция возвращает элемент <div>. Если внутри HTML-элемента необходимо вызвать XQuery-блок, то этот блок также ограничивается фигурными скобками. Если бы вместо

```
<div>
{
  (local:ShowHeader()),
  (local:ShowList()),
  (local:ShowFooter())
}
</div>
```

мы написали без фигурных скобок

```
<div>
  (local:ShowHeader()),
  (local:ShowList()),
  (local:ShowFooter())
</div>
```

то содержимое элемента `<div>` было бы воспринято как текст, а не как фрагмент программы на XQuery.

Еще одним непривычным моментом является то, что последовательные конструкции (которые должны выполняться друг за другом) необходимо перечислять через запятую. Конструкции можно группировать с помощью круглых скобок. Например:

```
(local:ShowHeader()),
(local:ShowList()),
(local:ShowFooter())
```

или

```
local:ShowHeader(),
local:ShowList(),
local:ShowFooter()
```

или

```
(
    local:ShowHeader()
), (
    local:ShowList()
), (
    local:ShowFooter()
)
```

далее мы будем достаточно часто использовать этот способ.

Вместо вызова функций здесь могли быть использованы любые конструкции XQuery, например, IF или FLWOR.

Таким же образом можно группировать тэги, например, мы могли бы определить следующую функцию:

```
declare function local:taggroup() as node()*
{
    <hr/>,
    <hr/>,
    <hr/>
};
```

Эта функция формирует на выходе три элемента «<hr/>». Тип возвращаемого значения этой функции «node()\*» – множество узлов. Если бы в качестве типа возвращаемого значения мы указали «node()», то получили бы сообщение об ошибке (несоответствие кардинальности), так как функция возвращает множество узлов (три элемента), а в соответствии с объявлением «node()» должна была бы возвращать один элемент.

Рассмотрим другие локальные функции.

```
(: ++++++ :)
```

```
(: Вывод данных в виде списка :)
```

```
(: ++++++ :)
```

```
declare function local:ShowList() as node()
```

Локальная функция, возвращает один узел.

```
{
```

```
<div>
```

```
  <table align="center" border="1">
```

```
    Формирование таблицы.
```

```
    <tr>
```

```
      <th>Тип процессора</th>
```

```
      <th colspan="2">Действия</th>
```

```
    </tr>
```

Первая строка таблицы определяет заголовок. Остальные строки таблицы формируются с помощью конструкции FLWOR:

```
{
```

```
  let $ProcessorsDoc := doc(xrx_example:ProcessorsFileName())
```

В переменную \$ProcessorsDoc сохраняется XML-документ, содержащий данные о процессорах. Библиотечная функция xrx\_example:ProcessorsFileName() возвращает путь и название документа, стандартная функция doc() по пути и названию читает документ из БД.

```
  for $Processor in $ProcessorsDoc//processor
```



Перебор всех элементов `processor` в документе. Здесь используется «`//`», а не «`/`», так как мы читаем не корневой элемент, а более глубоко вложенные элементы.

```
let
  $id := xs:string($Processor/@id),
  $nm := xs:string($Processor/processor_name)
```

Для каждого найденного процессора создаются связанные переменные (`bind-переменные`), содержащие `id` и наименование процессора.

```
order by $nm
```

Сортировка по возрастанию по наименованию процессора.

```
return
<tr>
  <td>{$nm}</td>
  {xrx_example:PathURI_edit_id_form($module_id, $module_ver,
  $id)}
  {xrx_example:PathURI_delete_id_form($module_id, $module_ver,
  $id)}
</tr>
}
```

Для каждого процессора возвращается строка HTML-таблицы (`<tr>`), содержащая три ячейки (`<td>`): первая содержит наименование процессора, вторая и третья содержат кнопки редактирования процессора и удаления процессора и формируются с помощью библиотечных функций. Для второй и третьей ячеек элементы `<td>` создаются внутри функций.

Каждая функция принимает в качестве параметров название и версию модуля (`$module_id` и `$module_ver`), которые нужны для формирования ссылок на другие XQuery-сценарии модуля, и параметр `$id`, содержащий `id` процессора для редактирования или удаления.

```
</table>
</div>
};
```

```
(: ++++++ :)
```

(: Верхний заголовок :)

```
(: ++++++ :)
```

```
declare function local:ShowHeader() as node()
{
<div>
  <table>
    <tr>
      {xrx_example:PathURI_menu()}
      {xrx_example:PathURI_edit_new_form($module_id, $module_ver)}
    </tr>
  </table>
  <hr/>
</div>
};
```

Функция верхнего заголовка формирует две кнопки: кнопку возврата в главное меню (`xrx_example:PathURI_menu()`) и кнопку добавления нового элемента (`xrx_example:PathURI_edit_new_form($module_id, $module_ver)`), а также горизонтальный разделитель (`<hr/>`).

```
(: ++++++ :)
```

(: Нижний заголовок :)

```
(: ++++++ :)
```

```
declare function local:ShowFooter() as node()
{
<div/>
};
```

Функция нижнего заголовка используется в качестве предварительной заготовки и возвращает пустой элемент `<div/>`.

#### 7.3.5.4.2 Сценарий `edit_processor_1.2.xql`

Сценарий предназначен для вывода формы редактирования названия процессора и кнопок сохранения и отмены редактирования.

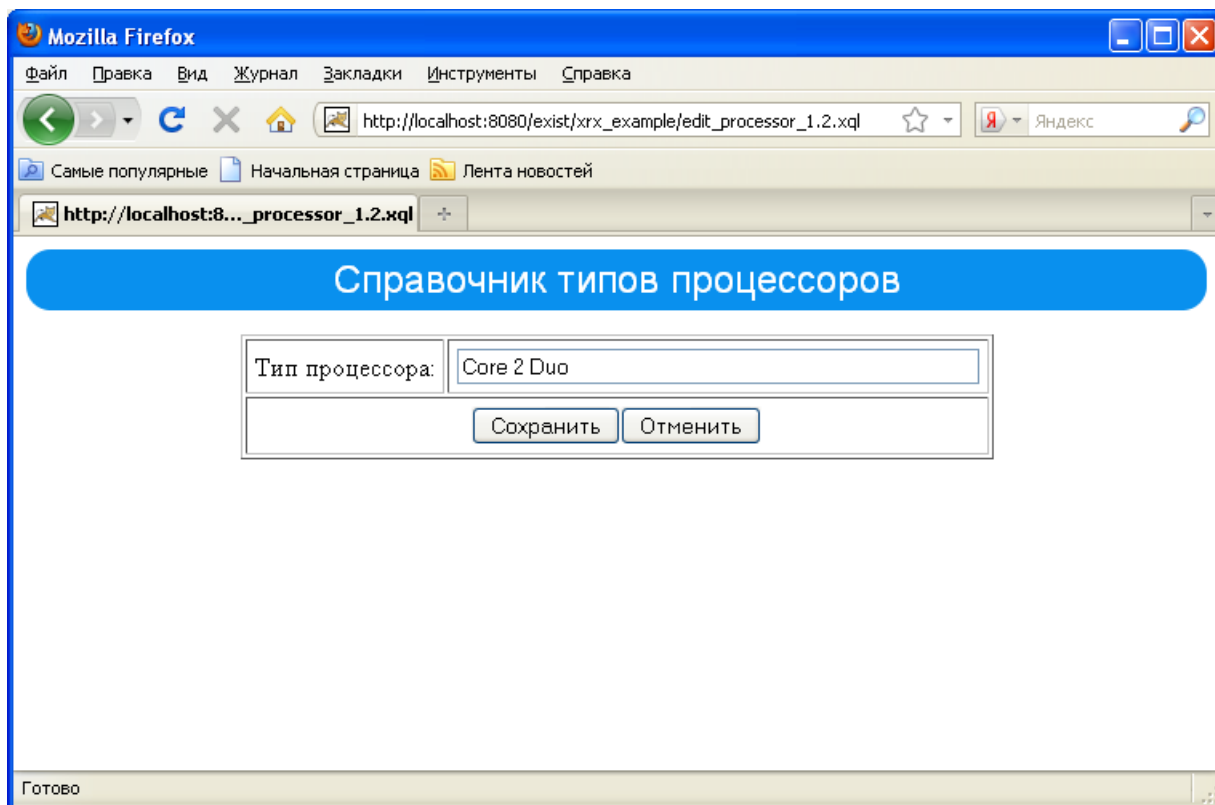


Рис. 7.7. Сценарий edit\_processor\_1.2.xql

Пролог сценария такой же, как в предыдущем случае:

```
xquery version "1.0";
(: Импорт стандартных модулей :)
declare namespace request="http://exist-db.org/xquery/request";
declare namespace session="http://exist-db.org/xquery/session";
(: Импорт модулей приложения :)
import module namespace xrx_example =
"http://iu5.bmstu.ru/edu/xrx_example" at "module.xqm";
(: Формат, в котором результат должен отображаться в браузере :)
declare option exist:serialize "method=xhtml media-type=text/html
indent=no";
(: Идентификатор модуля :)
declare variable $module_id {"processor"};
(: Версия модуля :)
declare variable $module_ver {"1.2"};
```

Основной запрос сценария:

```
(: ++++++ :)
```

```
(: Основной запрос :)
```

```
(: ++++++ :)
```

```
let
```

```
$newParam := xs:string(request:get-parameter("new","0")),
```

```
$idParam := xs:string(request:get-parameter("id","0")),
```

Функция `request:get-parameter` читает параметр HTTP-запроса. Параметр может быть задан в строке URL или передан из HTML-формы (читаются параметры, переданные и с помощью метода GET, и с помощью метода POST протокола HTTP). Первый аргумент функции `request:get-parameter` – это имя HTTP-параметра, второй аргумент – значение по умолчанию, если параметр не передан на сервер. Например, в случае `request:get-parameter("new","0")` функция пытается найти HTTP-параметр "new" и вернуть его значение. Если параметр не был передан, то возвращается "0".

```
$DataSaveURI := xrx_example:PathURI_data_save($module_id,
```

```
$module_ver),
```

```
$DataListURI := xrx_example:PathURI_list($module_id, $module_ver),
```

Формирование URI (имен) XQuery-сценариев для сохранения данных и отображения списка данных, сохранение этих URI в соответствующие переменные.

```
$IdParam := xrx_example:PathURI_make_id_param($module_id, $idParam,
```

```
$newParam),
```

```
$NameParam := local:make_name_param($idParam, $newParam)
```

Переменные `$IdParam` (id процессора) и `$NameParam` (наименование процессора) содержат данные по умолчанию для редактирования в HTML-форме. Эти переменные формируются с помощью вспомогательных функций.

```
return
```

```
<html>
```

```
  <head>
```

```
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-
```

```
8"/>
```

```
    <style type="text/css">
```

```

    <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
href="{xrx_example:PathURI_current_css()}" />
  </style>

```

**Вставка стиля CSS с помощью XInclude, как в предыдущем сценарии.**

```

</head>
<body>
  <h1>Справочник типов процессоров</h1>
  { local:main($DataListURI, $DataSaveURI, $IdParam, $NameParam) }
  Вызов функции local:main(), выполняющей основные действия.
</body>
</html>
(: ++++++ : )

```

**Локальные функции (они располагаются между прологом и основным запросом):**

```

(: ++++++ : )
(: Форма редактирования :)
(: ++++++ : )
declare function local:ShowForm($DataListURI as xs:string,
$DataSaveURI as xs:string, $IdParam as xs:string, $NameParam as
xs:string) as node()
{
<div>
<form method="post" action="{ $DataSaveURI }">

```

**Форма редактирования данных. Результаты заполнения формы будут отправлены серверному сценарию, название (URI) которого хранится в переменной \$DataSaveURI.**

```

<table align="center" border="1">
  <tr>
    <td>Тип процессора:</td>
    <td>
      <input type="hidden" name="id" value="{ $IdParam }" />
      <input type="text" size="50" name="processor_name"
value="{ $NameParam }" />

```

id процессора является скрытым полем (input type="hidden"), а наименование процессора полем ввода (input type="text").

Значения переменных \$IdParam и \$NameParam подставляются в форму в качестве значений по умолчанию.

То есть, когда HTML-форма будет сгенерирована, то в тэги формы будут вставлены значения по умолчанию в виде атрибутов value.

Если бы мы использовали технологию XForms, то значения по умолчанию не вставлялись бы в саму форму, но загружались бы из отдельного XQuery-сценария.

```

        </td>
    </tr>
    <tr>
        <td colspan="2" align="center">
            <button type="submit">Сохранить</button>
            <button type="button"
onclick="window.location='{ $DataListURI }'"> Отменить </button>
        </td>
    </tr>

```

В случае сохранения формы срабатывает стандартная кнопка типа «submit», которая отправляет форму серверному сценарию, адрес которого указан в атрибуте action тэга form.

В случае отмены редактирования, с использованием простого JavaScript-обработчика "window.location='{ \$DataListURI }'" мы загружаем в текущее окно браузера XQuery-сценарий, который отображает список процессоров (адрес этого сценария хранится в переменной \$DataListURI).

```

    </table>
</form>
</div>
};

```

```

(: ++++++ : )
(: Возвращает наименование для формы редактирования : )
(: ++++++ : )

```

```

declare function local:make_name_param($idParam as xs:string,
$newParam as xs:string) as xs:string
{
  if($idParam != "0")
    then
      (xs:string(doc(xrx_example:ProcessorsFileName())//processor[@id=$idP
aram]/processor_name))
    else (xs:string(""))
};
(: ++++++ :)
```

Функция принимает два строковых параметра: \$idParam (id элемента для редактирования) и \$newParam (признак нового элемента).

В этой функции в качестве основной XQuery-конструкции используется конструкция IF.

Если \$idParam не содержит значение «0», то это режим редактирования. Тогда в XML-документе, содержащем процессоры, производится поиск процессора с заданным id и возвращается название процессора. Иначе это режим добавления нового элемента, и возвращается пустая строка.

Параметр \$newParam в текущей версии функции не используется, но в случае усложнения алгоритма проверки он, возможно, потребуется.

Остальные функции сценария достаточно просты и не требуют пояснений:

```

(: ++++++ :)
```

(: Верхний заголовок :)

```

(: ++++++ :)
```

```

declare function local:ShowHeader() as node()
{
  <div/>
};

(: ++++++ :)
```

(: Нижний заголовок :)

```

(: ++++++ :)
```

```

declare function local:ShowFooter() as node()
{
<div/>
};

(: ++++++ : )
(: Основная функция :)
(: ++++++ : )
declare function local:main($DataListURI as xs:string, $DataSaveURI
as xs:string, $IdParam as xs:string, $NameParam as xs:string) as
node()
{
<div>
{
    (local:ShowHeader()),
    (local:ShowForm($DataListURI, $DataSaveURI, $IdParam,
$NameParam)),
    (local:ShowFooter())
}
</div>
};

(: ++++++ : )

```

#### 7.3.5.4.3 Сценарий data\_processor\_1.2.xql

Сценарий предназначен для работы с данными.

Пролог сценария стандартный. Дополнительно содержит переменные \$module\_computer\_id и \$module\_computer\_ver для взаимодействия с модулем компьютера:

```

xquery version "1.0";
(: Импорт стандартных модулей :)
declare namespace request="http://exist-db.org/xquery/request";
declare namespace response="http://exist-db.org/xquery/response";
declare namespace session="http://exist-db.org/xquery/session";
(: Импорт модулей приложения :)

```



```

import module namespace xrx_example =
"http://iu5.bmstu.ru/edu/xrx_example" at "module.xqm";
(: Формат в котором результат должен отображаться в браузере :)
declare option exist:serialize "method=xhtml media-type=text/html
indent=no";
(: Идентификатор модуля :)
declare variable $module_id {"processor"};
(: Версия модуля :)
declare variable $module_ver {"1.2"};
(: Для вызова модуля компьютера :)
declare variable $module_computer_id {"computer"};
declare variable $module_computer_ver {"1.1"};

```

Основной запрос сценария не формирует HTML-документ, как в предыдущих случаях. В основном запросе читаются возможные HTTP-параметры, по ним производится ветвление с помощью вложенных операторов IF и вызываются необходимые функции обработки данных (добавление, удаление и т.д.).

```

(: ++++++ : )
(: Основной запрос :)
(: ++++++ : )
let
  $newParam := xs:string(request:get-parameter("new","0")),
  $saveParam := xs:string(request:get-parameter("save","0")),
  $idParam := xs:string(request:get-parameter("id","0")),
  $nameParam := xs:string(request:get-
parameter("processor_name","0")),
  $deleteidParam := xs:string(request:get-
parameter("deleteid","0")),
  $listParam := xs:string(request:get-parameter("list","0")),
  $DataListURI := xrx_example:PathURI_list($module_id, $module_ver)
  Чтение HTTP-параметров и сохранение во временные переменные.
return
  if($saveParam = '1')

```

```
(: Сохранение данных :)
then
(
  local:SaveData($idParam, $nameParam, $DataListURI)
)
```

Если был передан HTTP-параметр «save=1», то читаются HTTP-параметры, переданные из HTML-формы редактирования «id=id процессора» и «processor\_name=наименование процессора» и вызывается функция сохранения данных.

```
else if($deleteidParam != '0')
(: Удаление данных :)
then
(
  local:DeleteData($deleteidParam, $DataListURI)
)
```

Если был передан HTTP-параметр «deleteid=id удаляемого процессора», то вызывается функция удаления данных.

```
else if($listParam = '1')
(: Выборка данных из справочника с сортировкой по наименованию :)
then
(
  for $data in doc(xrx_example:ProcessorsFileName())//processor
  order by $data/processor_name
  return $data
)
```

Если был передан HTTP-параметр «list=1», то процессоры возвращаются в виде множества элементов processor с сортировкой по наименованию процессора. Этот вызов используется для получения списка процессоров в модуле компьютера.

```
else ()
(: ++++++ :)
```

Локальные функции:

[Оглавление](#)

```
(: ++++++ :)
```

(: Создание элемента данных для процессора :)

```
(: ++++++ :)
```

```
declare function local:NewData($idParam as xs:string, $nameParam as
xs:string) as node()
{
<processor id="{ $idParam }">
  <processor_name>{ $nameParam }</processor_name>
</processor>
};
```

Функция формирует XML-элемент, соответствующий процессору, на основе переданных параметров: id процессора и наименование процессора.

```
(: ++++++ :)
```

(: Сообщения об ошибках :)

```
(: ++++++ :)
```

```
declare function local:CheckErrors($nameParam as xs:string) as
node() *
{
if(string-length(normalize-space($nameParam)) = 0)
  then (<p>Наименование не может быть пустой строкой</p>)
  else ()
};
```

Если название процессора не было введено, то функция возвращает сообщение об ошибке в виде элемента <p>, иначе возвращает пустое множество узлов.

```
(: ++++++ :)
```

(: Сохранение данных :)

```
(: ++++++ :)
```

```
declare function local:SaveData($idParam as xs:string, $nameParam as
xs:string, $DataListURI as xs:string) as node() *
{
let
  $Errors := local:CheckErrors($nameParam)
return
```

```

if(count($Errors) != 0)
(: Сообщения об ошибках :)
then
(
  xrx_example:ShowErrorReturnForm($module_id, $module_ver, "Ошибка
при сохранении данных", $Errors)
)

```

В переменную \$Errors сохраняются сообщения об ошибках. Если количество ошибок больше нуля, то выводится сообщение об ошибках с помощью функции `xrx_example:ShowErrorReturnForm()`.

```

(: Сохранение данных :)
else
(
  (
    Если ошибок не выявлено, то производится сохранение данных.
    if($idParam = 'new')
    then
    (: Добавление данных :)
    (
      let
        $newDataId := xrx_example:MakeUniqueName(),
        $newData:= local:NewData($newDataId, $nameParam)
      return
        update insert $newData into
doc(xrx_example:ProcessorsFileName())//processors
    )
  )

```

Если добавляются новые данные, то с помощью функции `xrx_example:MakeUniqueName()` генерируется уникальный идентификатор нового элемента и сохраняется в переменную \$newDataId. С помощью функции `local:NewData` генерируется новый элемент processor и сохраняется в переменную \$newData. Далее с помощью команды добавления данных «`update insert $newData into doc(xrx_example:ProcessorsFileName())//processors`» производится добавление сгенерированного элемента (который хранится в переменной \$newData) в XML-

файл, содержащий информацию о процессорах. Функция `xrx_example:ProcessorsFileName()` возвращает название файла с информацией о процессорах, функция `doc()` загружает файл, после этого в файле осуществляется поиск элемента `//processors`, в который и добавляется новый элемент `processor` из переменной `$newData`.

Конструкция «`update insert`» (несколько непривычная с точки зрения SQL) добавляет один XML-элемент внутрь другого XML-элемента.

```
else
  (: Обновление данных :)
  (
    let
      $newData:= local:NewData($idParam, $nameParam)
    return
      update replace
doc(xrx_example:ProcessorsFileName())//processor[@id=$idParam] with
$newData
```

Если данные о процессоре уже существовали, то с помощью команды «`update replace`» существующий элемент `processor` замещается новым элементом.

С помощью этой команды можно было бы замещать только название процессора. Но для того, чтобы сделать добавление и редактирование «почти одинаковыми» действиями, в обоих случаях элемент `processor` генерируется заново с помощью функции `local:NewData` (на основе `id` процессора и названия процессора) и в одном случае добавляется в XML-файл, а в другом случае замещает существующий в XML-файле элемент `processor`.

```
)
), (
  (: После успешного добавления или обновления данных переход к
списку :)
  xrx_example:redirect($DataListURI)
```

После успешного добавления или обновления данных с помощью функции `xrx_example:redirect()` осуществляется перенаправление на XQuery-сценарий, реализующий вывод списка процессоров.

При этом у пользователя возникает ощущение, что он после редактирования данных вернулся в список данных, в котором отображается отредактированное значение. Промежуточный вызов сценария сохранения данных для пользователя незаметен.

```
)
)
};
```

Перед удалением процессора из справочника необходимо проверить, что он не используется при описании компьютеров.

```
(: ++++++ : )
(: Сообщения об ошибках :)
(: ++++++ : )
declare function local:CheckDeleteErrors($deleteidParam as
xs:string) as node() *
{
(: Проверка того, что удаляемый тип процессора не используется в
компьютерах :)
let
  $ComputersList :=
collection(xrx_example:PathURI_data_list_in_db($module_computer_id))
,
  $cnt := count($ComputersList//computer[processor_id =
$deleteidParam])
```

Определение количества компьютеров, в которых используется удаляемый процессор.

```
return
  if($cnt > 0)
    then (<p>Этот тип процессора не может быть удален, так как он
используется при описании {$cnt} компьютера(ов)</p>)
    else ()
  Если это количество больше 0, то формируется сообщение об ошибке.
};
```

```

(: ++++++ : )
(: Удаление данных :)
(: ++++++ : )
declare function local:DeleteData($deleteidParam as xs:string,
$DataListURI as xs:string) as node()*
{
let
  $Errors := local:CheckDeleteErrors($deleteidParam)
return
  if(count($Errors) != 0)
  (: Сообщения об ошибках :)
  then
  (
    xrx_example:ShowErrorReturnList($module_id, $module_ver, "Ошибка
при удалении данных", $Errors)
  )

```

Если процессор используется при описании компьютеров, то его нельзя удалять, выводится сообщение об ошибке.

```

(: Удаление данных :)
else
(
  (
    let $del :=
doc(xrx_example:ProcessorsFileName())//processor[@id=$deleteidParam]
    return update delete $del

```

Иначе с помощью команды «update delete» элемент processor удаляется из XML-файла.

```

), (
(: После удаления данных переход к списку :)
xrx_example:redirect($DataListURI)

```

После этого осуществляется перенаправление на XQuery-сценарий, реализующий вывод списка процессоров.

```

)
)

```

} ;

#### **7.3.5.4.4 Взаимодействие между сценариями справочника «Тип процессора»**

Взаимодействие между сценариями справочника покажем с помощью диаграммы состояний (statechart diagram) в нотации UML.

Состоянию соответствует XQuery-сценарий, переходу – HTTP-запрос.

Здесь предполагается, например, что XQuery-сценарий, отображающий список данных, полностью отработает, выведет в браузер сгенерированный HTML-документ, который будет содержать кнопку добавления нового элемента. Если пользователь нажимает эту кнопку, то происходит HTTP-запрос к сценарию добавления (редактирования) данных. Этому HTTP-запросу соответствует переход из состояния «список данных» в состояние «редактирование данных».

Над стрелкой перехода указано выполняемое действие и параметры HTTP-запроса.



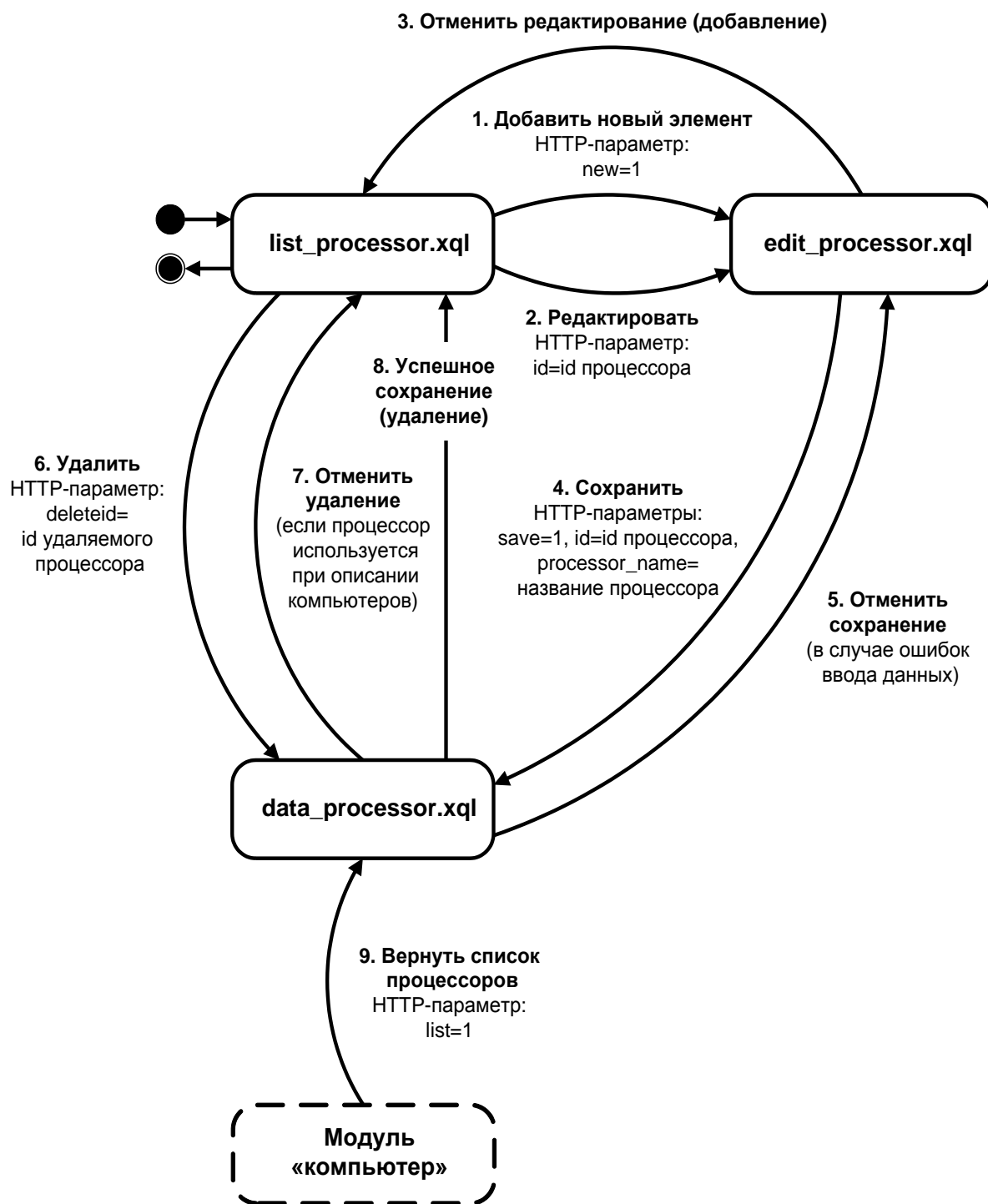


Рис. 7.8. Взаимодействие между сценариями справочника «Тип процессора».

Переход из начального состояния диаграммы осуществляется на сценарий «list\_processor.xql», так как пользователь начинает работу со справочником процессоров со списка данных.

Также из этого сценария осуществляется переход на конечное состояние. Этому переходу соответствует кнопка «Вернуться в главное меню».

В случае нажатия кнопки добавления нового элемента (1) осуществляется вызов сценария редактирования элемента «edit\_processor.xml». При этом передается HTTP-параметр «new=1». Получив этот параметр, сценарий «edit\_processor.xml» отображает пустую форму ввода данных.

В случае нажатия кнопки редактирования элемента (2) также осуществляется вызов сценария редактирования элемента «edit\_processor.xml». При этом передается HTTP-параметр «id=id редактируемого процессора». Получив этот параметр, сценарий «edit\_processor.xml» на основе параметра id читает данные из БД и отображает форму редактирования данных, которая заполнена данными редактируемого процессора.

В случае нажатия в форме редактирования (добавления) кнопки «Отменить» (3) осуществляется возврат в список данных.

В случае нажатия в форме редактирования (добавления) кнопки «Сохранить» (4) осуществляется вызов сценария для работы с данными «data\_processor.xml». Ему передаются следующие параметры HTTP-запроса: «save=1» – признак сохранения данных, «id=id сохраняемого процессора» (в случае добавления нового элемента вместо id сохраняемого процессора передается значение «new», это является признаком добавления данных), «processor\_name=название сохраняемого процессора».

Если данные введены неправильно (например, не задано название процессора), то выводится сообщение об ошибке и происходит возврат в форму редактирования (5).

Если данные введены правильно, то после сохранения данных в БД осуществляется переход в форму списка (8).

Если в форме списка «list\_processor.xml» нажата кнопка удаления данных (6), то осуществляется вызов сценария «data\_processor.xml». Ему передается параметр HTTP-запроса «deleteid=id удаляемого процессора». Далее проверяется возможность удаления данных. Если процессор с указанным id используется при описании компьютеров, то выводится сообщение об ошибке и происходит возврат

в форму списка (7). Если ошибок нет, то после удаления данных также осуществляется переход в форму списка (8).

Сценарий «list\_processor.xql» также может вызываться из сценариев модуля «компьютер» с параметром «list=1» для получения списка процессоров (9).

### 7.3.5.5 Справочник «Компьютер»

Так как справочник содержит несколько полей ввода, то будем использовать вариант архитектуры II, то есть технологию XForms.

Для реализации справочника необходимо разработать три сценария:

- list\_computer\_1.1.xql – список компьютеров (вид).
- edit\_computer\_1.1.xql – редактирование данных о компьютере (вид).
- data\_computer\_1.1.xql – редактирование данных (контроллер).

#### 7.3.5.5.1 Сценарий list\_computer\_1.1.xql

Сценарий предназначен для вывода списка компьютеров и кнопок добавления, редактирования, удаления. По структуре он очень похож на рассмотренный ранее сценарий вывода списка процессоров.

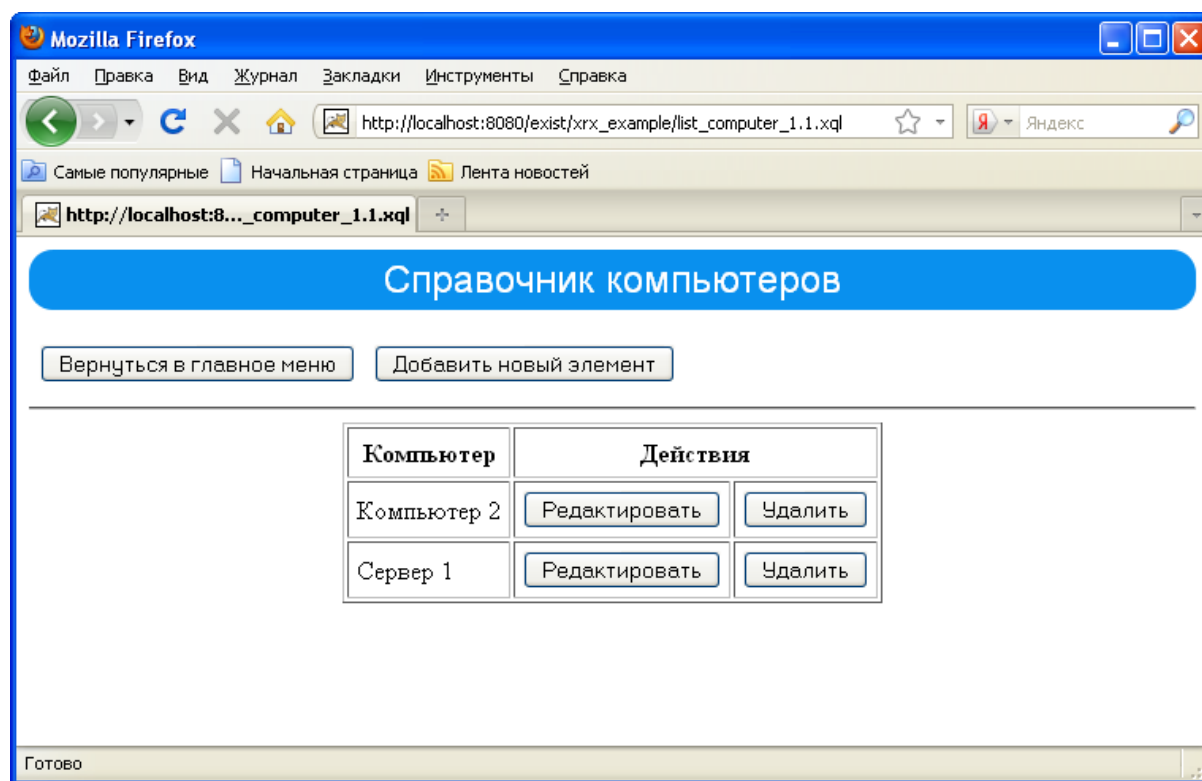


Рис. 7.9. Сценарий list\_computer\_1.1.xql

Текст сценария:

```
xquery version "1.0";

(: Импорт стандартных модулей :)

declare namespace request="http://exist-db.org/xquery/request";
declare namespace session="http://exist-db.org/xquery/session";

(: Импорт модулей приложения :)

import module namespace xrx_example =
"http://iu5.bmstu.ru/edu/xrx_example" at "module.xqm";

(: Формат в котором результат должен отображаться в браузере :)

declare option exist:serialize "method=xhtml media-type=text/html
indent=no";

(: Идентификатор модуля :)

declare variable $module_id {"computer"};

(: Версия модуля :)

declare variable $module_ver {"1.1"};
```

Переменные, содержащие идентификатор и версию для модуля компьютера.

```
(: ++++++ :)
```

```
(: Вывод данных в виде списка :)
```

```
(: ++++++ :)
```

```
declare function local:ShowList() as node()
```

```
{
```

```
<div>
```

```
  <table align="center" border="1">
```

```
    <tr>
```

```
      <th>Компьютер</th>
```

```
      <th colspan="2">Действия</th>
```

```
</tr>
```

```
{
```

```
  let $ComputersList :=
```

```
collection(xrx_example:PathURI_data_list_in_db($module_id))
```

```
  for $Computer in $ComputersList//computer
```

```

let
    $id := xs:string($Computer/@id),
    $ComputerNm := xs:string($Computer/computer_name)
order by $ComputerNm
return
<tr>
    <td>{$ComputerNm}</td>
    {xrx_example:PathURI_edit_id_form($module_id, $module_ver,
$id)}
    {xrx_example:PathURI_delete_id_form($module_id,
$module_ver, $id)}
</tr>
}
</table>
</div>
};
(: ++++++ : )
(: Верхний заголовок :)
(: ++++++ : )
declare function local:ShowHeader() as node()
{
<div>
    <table>
        <tr>
            {xrx_example:PathURI_menu()}
            {xrx_example:PathURI_edit_new_form($module_id, $module_ver)}
        </tr>
    </table>
    <hr/>
</div>
};
(: ++++++ : )
(: Нижний заголовок :)
(: ++++++ : )
declare function local:ShowFooter() as node()

```

```

{
<div/>
};
(: ++++++ : )
(: Основная функция :)
(: ++++++ : )
declare function local:main() as node()
{
<div>
  {
    (local:ShowHeader()),
    (local:ShowList()),
    (local:ShowFooter())
  }
</div>
};
(: ++++++ : )
(: Основной запрос :)
(: ++++++ : )
let $CurrentCssURI := xrx_example:PathURI_current_css()
return
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8"/>
    <style type="text/css">
      <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
href="{ $CurrentCssURI }"/>
    </style>
  </head>
  <body>
    <h1>Справочник компьютеров</h1>
    { local:main() }
  </body>
</html>

```

(: ++++++ :)

### 7.3.5.5.2 Сценарий edit\_computer\_1.1.xql

Сценарий предназначен для вывода формы редактирования компьютера и кнопок сохранения и отмены редактирования. Сценарий генерирует XHTML-документ, в котором находится XForms -форма.

The screenshot shows a web browser window with the following content:

- Title Bar:** Mozilla Firefox
- Address Bar:** http://localhost:8080/exist/xrx\_example/edit\_computer\_1.1.xql
- Page Title:** Справочник компьютеров
- Form Fields:**
  - Наименование компьютера: Сервер 1 \*
  - Тип процессора: Core 2 Duo \*
  - Объем ОП: (в Мб): 1024 \*, (в Гб): 1 [без округления: 1]
  - Объем жесткого диска (в Гб): 1000 \*
  - Используется как сервер: ☒
- Список модернизаций (List of upgrades):**

Дата	Описание	Действие
04/10/2010	увеличение ОП	Удалить
05/10/2010	замена диска	Удалить
- Buttons:** Добавить, Сохранить, Отменить
- Status Bar:** Готово

Рис. 7.10. Сценарий edit\_computer\_1.1.xql

Пролог сценария:

```
xquery version "1.0";
```

(: Импорт стандартных модулей :)

```
declare namespace request="http://exist-db.org/xquery/request";
```

```

declare namespace session="http://exist-db.org/xquery/session";
(: Импорт модулей приложения :)
import module namespace xrx_example =
"http://iu5.bmstu.ru/edu/xrx_example" at "module.xqm";
(: Формат, в котором результат должен отображаться в браузере :)
declare option exist:serialize "method=xhtml media-type=text/html
indent=no";
(: Идентификатор модуля :)
declare variable $module_id {"computer"};
(: Версия модуля :)
declare variable $module_ver {"1.1"};
(: Для вызова модуля процессора :)
declare variable $module_processor_id {"processor"};
declare variable $module_processor_ver {"1.2"};

```

Пролог содержит переменные \$module\_processor\_id и \$module\_processor\_ver для взаимодействия с модулем процессора.

### Основной запрос сценария:

```

(: ++++++ : )
(: Основной запрос :)
(: ++++++ : )
(: Если не заполнен справочник типов процессоров, то редактирование
компьютеров невозможно (нет данных для выпадающего списка
процессоров) :)
if (xrx_example:ModuleDataCount($module_processor_id,
$module_processor_ver) = 0)
then
  (: Вывод сообщения об ошибке :)
  (
    xrx_example:ShowErrorButton("Невозможно редактировать данные о
компьютере пока не заполнен справочник типов процессоров",
"Вернуться в главное меню", $xrx_example:MainScript)
  )

```



Перед генерацией формы ввода данных о компьютере необходимо проверить, что существуют данные в справочнике типов процессоров. Если этот справочник пустой, то отсутствуют данные для выпадающего списка типов процессоров и это приведет к возникновению ошибки в XForms-процессоре.

Поэтому, если справочник типов процессоров не заполнен, то выводится сообщение об ошибке и кнопка возврата, иначе выполняются основные действия по генерации формы.

```
else
(: Выполнение основных действий :)
(
let
$newParam := xs:string(request:get-parameter("new","0")),
$idParam := request:get-parameter("id","0"),
$DataReadURI := xrx_example:PathURI_data_edit($module_id,
$module_ver, $idParam, $newParam),
$DataPreSaveURI := xrx_example:PathURI_data_presave($module_id,
$module_ver),
$DataSaveURI := xrx_example:PathURI_data_save($module_id,
$module_ver),
$ListURI := xrx_example:PathURI_list($module_id, $module_ver),
$ProcessorReadURI :=
xrx_example:PathURI_data_list($module_processor_id,
$module_processor_ver),
$DataNewUpgradeURI := concat(xrx_example:PathURI_data($module_id,
$module_ver), "?newupgrade=1"),
$CurrentCssURI := xrx_example:PathURI_current_css(),
$XFormsCssURI := xrx_example:PathURI_xforms_css(),
```

Чтение параметров и формирование вспомогательных URI. Далее формируется XHTML-документ, содержащий XForms-форму.

```
$form :=
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events">
```

```

<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8"/>
  <style type="text/css">
    <!-- Добавление стилей CSS с помощью XInclude -->
    <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
href="{ $CurrentCssURI }"/>
    <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
href="{ $XFormsCssURI }"/>
  </style>
  <!-- Модель создается в секции head -->
  { local:CreateXFormsModels($DataReadURI, $DataSaveURI,
$ProcessorReadURI, $DataNewUpgradeURI, $DataPreSaveURI) }

```

Вызов функции, которая создает модели данных XForms-формы в секции HEAD XHTML-документа.

```

</head>
<body>
  <h1>Справочник компьютеров</h1>
  <!-- Поля редактирования формы -->
  { local:EditForm($ListURI, $DataPreSaveURI, $DataSaveURI) }

```

Вызов функции, которая создает поля ввода данных XForms-формы в секции BODY XHTML-документа.

```

</body>
</html>

(: сгенерированный XForms-документ обрабатывается с помощью XSLT-
преобразования для получения HTML-документа :)
return xrx_example:transform_xsltforms($form)
)

```

Сгенерированный XHTML-документ сохраняется во вспомогательную переменную \$form. Далее его необходимо преобразовать с использованием XSLT-преобразования, которое является основной частью XForms-процессора XSLTForms. Это преобразование можно было сделать в браузере, но здесь оно

делается в серверном сценарии и в браузер передается результат преобразования в виде HTML-документа.

В этом сценарии всего две локальных функции, которые генерируют модель данных формы и поля ввода формы:

```
(: ++++++ :)  
(: Модели данных формы :)  
(: ++++++ :)  
declare function local:CreateXFormsModels($DataReadURI as xs:string,  
$DataSaveURI as xs:string, $ProcessorReadURI as xs:string,  
$DataNewUpgradeURI as xs:string, $DataPreSaveURI as xs:string) as  
node() *  
{  
(  
(: Основная модель данных :)  
<xforms:model id="computer_model">
```

Основная модель данных «computer\_model» содержит информацию о добавляемом или редактируемом компьютере и его обновлениях.

```
<!-- Основные данные (о компьютере) -->  
<xforms:instance id="computer_instance" xmlns=""  
src="{ $DataReadURI }"/>
```

Инстанс формы «computer\_instance» содержит XML-элемент, соответствующий данным о компьютере. Данные читаются из XQuery-сценария, название которого хранится в переменной \$DataReadURI. В случае добавления данных в эту переменную помещается URI сценария, который возвращает незаполненный XML-элемент для ввода новых данных. В случае редактирования данных в эту переменную помещается URI сценария, который возвращает данные о компьютере из БД на основе id компьютера.

Атрибут «xmlns=""» означает, что пространства имен в данных не используются (используется «пустое» пространство имен).

```
<!-- Данные для добавления нового обновления -->
```

```
<xforms:instance id="upgrade_instance" xmlns=""
src="{ $DataNewUpgradeURI }"/>
```

Инстанс «upgrade\_instance» содержит XML-данные, соответствующие одной модернизации, этот инстанс используется при добавлении новой модернизации.

Далее следуют элементы xforms:bind, которые описывают ограничения для XML-элементов данных. Атрибут nodeset задает название элемента данных (в виде XPath-выражения), атрибут required определяет обязательные элементы (которые не могут быть пустыми), атрибут type задает тип данных элемента, атрибут constraint задает ограничение на значение данных.

Особенностью текущей версии XSLTForms является то, что для одного элемента данных можно определить только один элемент xforms:bind. Нельзя, например, задать для одного элемента данных два xforms:bind и в одном указать атрибут required, а в другом атрибут constraint, все атрибуты необходимо сгруппировать в одном элементе xforms:bind.

```
<!-- Наименование компьютера - обязательное поле и введенные
данные не состоят только из пробелов -->
<xforms:bind nodeset="//computer_name" required="true()"
constraint="string-length(normalize-space(.)) > 0" />
```

Атрибут required="true()" означает, что поле обязательно для ввода.

По умолчанию тип данных элемента – строка. Атрибут constraint задает дополнительные ограничения на значение данных. Функция normalize-space(.) удаляет все пробелы слева и справа в текущем элементе, функция string-length возвращает длину строки. Длина строки должна быть больше нуля, это означает, что строка содержит не только пробелы.

```
<!-- Тип процессора - обязательное поле -->
<xforms:bind nodeset="//processor_id" required="true()" />
<!-- ОП - обязательное поле целого типа в диапазоне от 0 до 4096 -
->
<xforms:bind nodeset="//op" type="xforms:int" required="true()"
constraint="(. >= 1) and (. <= 4096)" />
```

Атрибут `type="xforms:int"` означает, что это элемент целочисленного типа. В атрибуте `constraint="( . &gt;= 1) and ( . &lt;= 4096)"` проверяется, что значение текущего элемента (точка означает текущий элемент) должно лежать в диапазоне от 1 до 4096, «&gt;=» – это больше или равно (>=), «&lt;=» – это меньше или равно (<=).

```
<!-- Объем жесткого диска - обязательное поле целого типа в
диапазоне от 0 до 1000 -->
<xforms:bind nodeset="//hdd" type="xforms:int" required="true()"
constraint="( . &gt;= 1) and ( . &lt;= 1000)" />

<!-- Признак сервера - логического типа -->
<xforms:bind nodeset="//server" type="xforms:boolean" />

<!-- Дата модернизации - обязательное поле типа дата -->
<xforms:bind nodeset="//upgrade_date" type="xforms:date"
required="true()" />

<!-- Описание модернизации - обязательное поле и введенные данные
не состоят только из пробелов -->
<xforms:bind nodeset="//upgrade_description" required="true()"
constraint="string-length(normalize-space(.)) &gt; 0" />

<!-- Отправка формы -->
<xforms:submission id="submit_id" replace="none"
action="{ $DataPreSaveURI }" method="post"
includenamespaceprefixes="" />
```

Элемент `xforms:submission` определяет параметры сохранения заполненных данных формы.

Атрибут «`id="submit_id"`» определяет id элемента для дальнейших ссылок в форме.

Атрибут «`replace="none"`» определяет, что после сохранения данных форма не должна изменяться. В нашем случае более удобным было бы значение «`replace="all"`», при котором форма после сохранения заменяется результатом

работы серверного сценария (там может быть сообщение об успешном сохранении формы и кнопки перехода к другим формам), но этот режим не поддерживается в текущей версии XSLTForms.

Атрибут «action="{ \$DataPreSaveURI}"» задает URI серверного сценария, которому будут передаваться данные.

Атрибут «method="post"» определяет, что данные будут передаваться с помощью метода POST протокола HTTP.

Атрибут «includenamespacesprefixes=""» определяет, что в передаваемые данные не нужно включать никакие префиксы пространств имен (в нашем примере XML-данные не используют пространства имен).

```
</xforms:model>
), (
(: Данные для выбора типов процессоров :)
<xforms:model id="processor_model">
  <xforms:instance id="processor_instance" xmlns=""
src="{ $ProcessorReadURI}"/>
```

Модель «processor\_model» содержит инстанс «processor\_instance», в который загружается список типов процессоров. Переменная \$ProcessorReadURI содержит URI сценария, который возвращает список типов процессоров (в нашем случае это сценарий «data\_computer\_1.1.xql» с параметром «list=1»).

```
</xforms:model>
)
};

(: ++++++ : )
(: Редактирование формы :)
(: ++++++ : )
declare function local:EditForm($ListURI as xs:string,
$DataPreSaveURI as xs:string, $DataSaveURI as xs:string) as node()
{
<div>
  <table border="1" align="center" >
```

Элементы формы форматируются с помощью таблицы HTML.

```
<tr>
  <th colspan="2" align="right">Наименование компьютера:</th>
  <td>
    <xforms:input model="computer_model" ref="computer_name">
      <xforms:alert>Поле не может быть пустым</xforms:alert>
    </xforms:input>
  </td>
```

Наименование компьютера вводится с помощью элемента `xforms:input`. Атрибут `model="computer_model"` ссылается на модель данных, а атрибут `ref="computer_name"` на XML-элемент в модели данных.

Особенностью является то, что явной ссылки на инстанс здесь нет, хотя модель «computer\_model» содержит два инстанса. По умолчанию используется первый инстанс, поэтому приведенный пример должен работать корректно. Но если в модели поменять порядок следования инстансов, то приведенный пример работать не будет. Для явной ссылки на инстанс можно использовать XPath-функцию «instance» (она применяется в XPath-выражении в атрибуте `ref`), тогда атрибут `ref="instance('computer_instance')//computer_name"`. При использовании функции «instance» можно вводить данные в любой инстанс модели.

Элемент `xforms:alert` содержит сообщение об ошибке. Если данные вводятся неправильно (то есть нарушется какое-либо из условий, указанных в `xforms:bind` для данного элемента), то справа от поля ввода появляется пиктограмма ошибки и сообщение из `xforms:alert`. Поскольку в случае всех ошибок выдается одно и то же сообщение, то в элементе `xforms:alert`, вероятно, лучше не перечислять все возможные ошибки, а задавать подсказку для правильного ввода данных.

```
</td>
</tr>

<tr>
  <th colspan="2" align="right">Тип процессора:</th>
  <td>
    <xforms:select1 model="computer_model" ref="processor_id">
```

Данные о процессорах выбираются из выпадающего списка (xforms:select1). Атрибут model="computer\_model" ссылается на модель данных, а атрибут ref="processor\_id" задает элемент данных, в который будет сохранено значение из выбранного пункта списка.

```
<xforms:alert>Необходимо выбрать тип
процессора</xforms:alert>
<xforms:itemset model="processor_model"
nodeset="//processor">
  <xforms:value ref="@id"/>
  <xforms:label ref="processor_name"/>
</xforms:itemset>
```

Элемент xforms:itemset определяет пункты списка. Пункту списка соответствует элемент «processor» в первом инстансе модели данных «processor\_model».

Элемент xforms:value определяет данные пункта списка, которые будут записаны в результирующий элемент.

Элемент xforms:label определяет текст, отображаемый в пункте списка.

В нашем случае пункты списка читаются из первого инстанса модели данных «processor\_model». Пункту соответствует элемент «processor». В пунктах списка отображается значение элемента «processor\_name», вложенного в элемент «processor». При выборе пункта списка значение атрибута «id» выбранного элемента «processor» записывается в другую модель данных «computer\_model» в XML-элемент «processor\_id».

```
</xforms:select1>
</td>
</tr>

<tr>
  <th rowspan="2" align="right">Объем ОП:</th>
  <th align="right">(в Мб) :</th>
  <td>
    <xforms:input model="computer_model" ref="op">
```



```

        <xforms:alert>Значение должно быть целым числом в
диапазоне от 1 до 4096</xforms:alert>
    </xforms:input>

```

Поле «Объем ОП (в Мб)» вводится как целое число в заданном диапазоне. Эти ограничения указаны в модели данных в соответствующем элементе `xforms:bind`.

```

    </td>
</tr>

<tr>
    <th align="right">(в Гб):</th>
    <td>
        <xforms:output model="computer_model"
value="concat(string(round(op div 1024)), '[без округления: ',
string(op div 1024), ']')">
        <xforms:alert>Значение вычисляется на основе поля "Объем
ОП (в Мб)"</xforms:alert>
    </xforms:output>

```

Поле «Объем ОП (в Гб)» вычисляется на основе поля «Объем ОП (в Мб)» и не сохраняется в модель данных.

Формула для вычисления указана в атрибуте `value` элемента `xforms:output`: `(op div 1024)` – деление на 1024 для перевода Мб в Гб, функция `round` осуществляет округление до целого числа, функция `string` преобразует число в строку, функция `concat` соединяет строки с округленным и неокругленным значением.

Если необходимо сделать так, чтобы вычисляемое поле сохранялось в модель данных, то необходимо:

1. Ввести новый элемент в модель данных.
2. Для этого элемента определить `xforms:bind` с атрибутом `calculate`, значением атрибута `calculate` должна быть формула для вычисления.
3. В элементе `xforms:output` в атрибуте `value` задать только имя нового элемента, который уже вычислен в модели на основе `xforms:bind`.

```

    </td>
</tr>

```

```

<tr>
  <th colspan="2" align="right">Объем жесткого диска (в
Гб):</th>
  <td>
    <xforms:input model="computer_model" ref="hdd">
      <xforms:alert>Значение должно быть целым числом в
диапазоне от 1 до 1000</xforms:alert>
    </xforms:input>

```

Поле «Объем жесткого диска» вводится как целое число в заданном диапазоне.

```

    </td>
</tr>

```

```

<tr>
  <th colspan="2" align="right">Используется как сервер:</th>
  <td>
    <xforms:input model="computer_model" ref="server">
      <xforms:hint>Если компьютер используется в качестве
сервера, то флаг должен быть включен</xforms:hint>
    </xforms:input>

```

Поле «Используется как сервер» отображается в виде флажка, так как для него указан логический тип данных в xforms:bind.

```

    </td>
</tr>

```

```

<tr>
  <td colspan="3">
    <h2>Список модернизаций</h2>

    <ol>

```

```
<xforms:repeat id="repeat_upgrades" model="computer_model"
nodeset="//upgrades/upgrade[@id]" appearance="compact">
```

Список модернизаций отображается с помощью `xforms:repeat`.

Атрибут `nodeset="//upgrades/upgrade[@id]"` определяет в качестве повторяющегося элемента элемент «upgrade», который вложен в элемент «upgrades».

При этом, у элемента «upgrade» обязательно должен быть указан атрибут «id». Это условие задается потому, что по умолчанию в элемент «upgrades» вложен один пустой элемент «upgrade» (без этого неправильно работает добавление новых элементов «upgrade»).

Это решение является неправильным с точки зрения XForms, так как модель данных является первичной, а все элементы ввода данных должны под нее подстраиваться, но, к сожалению, более удачного решения здесь найти не удалось.

Атрибут `appearance="compact"` предписывает отображать вложенные элементы в одной строке.

```
<li>
```

```
<xforms:input model="computer_model"
ref="upgrade_date">
  <xforms:label>Дата:</xforms:label>
  <xforms:alert>Поле не может быть пустым и должно
содержать дату</xforms:alert>
</xforms:input>
```

Поле ввода даты модернизации.

```
<xforms:textarea model="computer_model"
ref="upgrade_description">
  <xforms:label>Описание:</xforms:label>
  <xforms:alert>Поле не может быть
пустым</xforms:alert>
</xforms:textarea>
```

Многострочное поле ввода описания модернизации.

```

<!-- Удаление модернизации -->
<xforms:trigger>
  <xforms:label>Удалить</xforms:label>
  <xforms:delete ev:event="DOMActivate"
model="computer_model" nodeset="." />
</xforms:trigger>

```

Кнопка удаления текущего элемента «upgrade». Элемент `xforms:delete` удаляет элемент из заданной модели (`model="computer_model"`). Удаляется текущий элемент «upgrade» (`nodeset="."`).

Событие `DOMActivate` возникает при нажатии на кнопку или при нажатии клавиши «ввод». Атрибут `ev:event="DOMActivate"` элемента `xforms:delete` указывает на то, что `xforms:delete` должен выполняться, если элемент `xforms:trigger` активирован.

Фактически атрибут `ev:event="DOMActivate"` означает, что элемент `xforms:delete` является обработчиком события по нажатию `xforms:trigger`.

```

</li>
</xforms:repeat>
</ol>

```

```

<!-- Добавление модернизации -->
<xforms:trigger>

```

Кнопка добавления данных о модернизации.

```

<xforms:label>Добавить</xforms:label>
<!-- Действия -->
<xforms:action ev:event="DOMActivate">

```

Несколько действий объединяются с помощью одного элемента верхнего уровня `xforms:action`, который является обработчиком события по нажатию кнопки (атрибут `ev:event="DOMActivate"`).

В `xforms:action` вложены команды, которые добавляют новый XML-элемент и устанавливают значение по умолчанию для добавленного элемента, эти команды выполняются последовательно, в указанном порядке.

```

<!-- Добавление элемента из отдельного instance -->

```

```
<xforms:insert model="computer_model"
nodeset="//upgrades/upgrade" at="last()" position="after"
origin="instance('upgrade_instance')//upgrade" />
```

Команда `xforms:insert` добавляет новый элемент «upgrade».

Атрибут `model="computer_model"` задает модель данных.

Следующие атрибуты задают приемник данных.

Атрибут `nodeset="//upgrades/upgrade"` указывает, что при добавлении нужно ориентироваться на элемент «upgrade», вложенный в «upgrades» (поэтому и приходится по умолчанию добавлять хотя бы один пустой «upgrade»), `at="last()"` – ориентироваться на последний элемент «upgrade», `position="after"` – добавлять после элемента «upgrade».

Источник данных задается атрибутом `origin`.

Атрибут `origin="instance('upgrade_instance')//upgrade"` указывает, что данные добавляются из элемента `upgrade`, который находится в другом инстансе модели.

Как только новый элемент будет добавлен, он сразу попадает в область действия приведенного выше `xforms:repeat` (то есть соответствует XPath-выражению, указанному в атрибуте `nodeset` элемента `xforms:repeat`) и отображается как строка с информацией о модернизации.

При этом сама кнопка добавления находится вне `xforms:repeat`.

```
<!-- Генерация id - id генерируется как номер элемента в
списке и текущие дата и время -->
<xforms:setvalue model="computer_model"
ref="//upgrade[last()]/@id"
value="concat(string(index('repeat_upgrades')), '_',
string(now()))"/>
```

С помощью `xforms:setvalue` устанавливается значение по умолчанию для элементов формы.

В нашем случае добавляемый элемент «upgrade» содержит пустые вложенные поля данных, поэтому явно очищать их с помощью `xforms:setvalue` не нужно.

С помощью `xforms:setvalue` мы генерируем уникальный атрибут «id» для нового элемента «upgrade».

Вообще, язык XPath содержит функцию `generate-id`, которая генерирует уникальный идентификатор для узла (возможно, в некоторых XForms-процессорах можно использовать эту функцию). Но в XSLTForms она не поддерживается, поэтому приходится использовать чуть более сложный способ.

Атрибут `ref="//upgrade[last()]/@id"` указывает, что данные будут сохраняться в последний элемент «upgrade» (функция `last()` возвращает номер последнего элемента), в атрибут «id».

Атрибут `value="concat(string(index('repeat_upgrades')), '_', string(now()))"` содержит вычисляемое значение для сохранения. Уникальный id определяется как конкатенация текущей даты и времени (функция `now()`) и номера нового элемента в `xforms:repeat` (функция `index` возвращает номер текущего элемента в `xforms:repeat`, аргументом является атрибут `id` элемента `xforms:repeat`).

Поскольку `now()` возвращает время с точностью до секунды, то значение потеряет уникальность, только если мы в течение секунды создадим элемент (например третий по счету, тогда `id=3_время`), удалим один из предыдущих элементов и снова создадим элемент (он также третий по счету, `id=3_время`), что маловероятно.

```

        </xforms:action>
    </xforms:trigger>

</td>
</tr>

<tr>
    <td colspan="3" align="center">

        <xforms:trigger>

```

Кнопка сохранения данных.

```
<xforms:label>Сохранить</xforms:label>
```

```
<xforms:action ev:event="DOMActivate" if="is-  
valid(instance('computer_instance'))">
```

Элемент `xforms:action` задает обработчик события при нажатии на кнопку. Атрибут `if="is-valid(instance('computer_instance'))"` означает, что вложенные в `xforms:action` действия будут выполнены, только если значение атрибута `if` (XPath-выражение) истинно.

Здесь проверяется, что данные в инстансе «`computer_instance`» (то есть данные в нашей форме) должны быть введены без ошибок. Функция `is-valid` возвращает истинное значение, только если все данные в инстансе заполнены правильно, то есть выполняются все ограничения на данные, заданные в элементах `xforms:bind`.

```
<!-- Данные передаются на сервер и сохраняются в  
сессионной переменной -->
```

```
<xforms:send submission="submit_id" />
```

Так как XSLTForms не поддерживает режим сохранения данных «`replace="all"`» (при котором форма после сохранения заменяется результатом работы серверного сценария), то приходится использовать «двухшаговое» сохранение данных.

Первый шаг состоит в отправке данных с помощью команды `xforms:send`, которая отправляет данные в соответствии с заданным элементом `submission`. С помощью атрибута `submission="submit_id"` мы ссылаемся на заданный в модели элемент `submission` с атрибутом `id="submit_id"`.

В нашем случае мы передаем данные серверному сценарию, который предварительно сохраняет их в сессионной переменной. Сессия уникальна в рамках соединения браузера с веб-приложением, поэтому использование сессионной переменной кажется наиболее правильным решением для временного хранения данных.

Это предварительное сохранение данных (`presave`).

Передача данных происходит в асинхронном режиме, форма остается в текущем окне браузера.

```

        <!-- Данные читаются из сессионной переменной,
        проверяются и сохраняются в БД -->
        <xforms:load show="replace" resource="{ $DataSaveURI}"/>

```

На втором шаге с помощью команды `xforms:load` мы загружаем в текущее окно результаты работы другого серверного сценария, заменяя этим форму в текущем окне браузера.

Этот сценарий читает предварительно сохраненные данные из сессионной переменной и осуществляет дополнительную проверку данных (которую, возможно, не удалось реализовать средствами XForms). В случае ошибок выводится сообщение об ошибках и кнопка возврата к форме редактирования, если ошибок нет, то данные сохраняются в БД.

```

        </xforms:action>
    </xforms:trigger>

    <xforms:trigger>
        <xforms:label>Отменить</xforms:label>
        <xforms:load ev:event="DOMActivate" show="replace"
resource="{ $ListURI}"/>

```

Кнопка осуществляет переход на сценарий, выводящий список данных, форма не сохраняется.

```

        </xforms:trigger>

    </td>
</tr>
</table>
</div>
};

```

#### 7.3.5.5.3 Сценарий `data_computer_1.1.xql`

Сценарий предназначен для работы с данными.

По структуре он очень похож на рассмотренный ранее сценарий для процессоров, есть только некоторые особенности, связанные с «двухшаговым» сохранением.



## Пролог сценария стандартный:

```
xquery version "1.0";

(: Импорт стандартных модулей :)

declare namespace request="http://exist-db.org/xquery/request";
declare namespace response="http://exist-db.org/xquery/response";
declare namespace session="http://exist-db.org/xquery/session";

(: Импорт модулей приложения :)

import module namespace xrx_example =
"http://iu5.bmstu.ru/edu/xrx_example" at "module.xqm";

(: Формат, в котором результат должен отображаться в браузере :)

declare option exist:serialize "method=xhtml media-type=text/html
indent=no";

(: Идентификатор модуля :)

declare variable $module_id {"computer"};

(: Версия модуля :)

declare variable $module_ver {"1.1"};

(: Название сессионной переменной для временного хранения данных
формы :)

declare variable $current data {"current data"};
```

В основном запросе читаются возможные HTTP-параметры, по ним производится ветвление с помощью вложенных операторов IF и вызываются необходимые функции обработки данных (добавление, удаление и т.д.):

```
(: ++++++ :)  
(: Основной запрос :)  
(: ++++++ :)  
let  
  $newParam := xs:string(request:get-parameter("new","0")),  
  $newUpgradeParam := xs:string(request:get-  
parameter("newupgrade","0")),  
  $saveParam := xs:string(request:get-parameter("save","0")),  
  $presaveParam := xs:string(request:get-parameter("presave","0")),  
  $idParam := xs:string(request:get-parameter("id","0")),
```

```

$nameParam := xs:string(request:get-
parameter("processor_name","0")),
$deleteidParam := xs:string(request:get-
parameter("deleteid","0")),
$DataListURI := xrx_example:PathURI_list($module_id, $module_ver)

```

**Чтение HTTP-параметров и сохранение во временные переменные.**

```

return
if($newParam = '1')
  (: Создание нового экземпляра :)
  then
  (
    let $newDataId := xrx_example:MakeUniqueName()
    return local:NewData($newDataId)
  )

```

Если был передан HTTP-параметр «new=1», то возвращается пустой XML-элемент «computer». Этот режим используется в форме при добавлении нового компьютера. Особенностью является то, что запрос с этим параметром отправляется из XForms-формы при инициализации данных.

```

else if($newUpgradeParam = '1')
  (: Создание нового экземпляра для upgrade :)
  then
  (
    let $NewUpgrade := local:NewUpgradeData()
    return $NewUpgrade
  )

```

Если был передан HTTP-параметр «newupgrade=1», то возвращается пустой XML-элемент «upgrade». Этот режим используется в XForms-форме при добавлении модернизации компьютера.

```

else if($saveParam = '1')
  (: Сохранение данных :)
  then
  (
    local:SaveData($DataListURI)
  )

```

Если был передан HTTP-параметр «save=1», то выполняется проверка и сохранение в БД данных, предварительно сохраненных в сессионной переменной.

```
else if($presaveParam = '1')
  (: Сохранение данных во временной переменной :)
  then
  (
    local:PreSaveData()
  )
```

Если был передан HTTP-параметр «presave=1», то данные предварительно сохраняются в сессионной переменной.

```
else if($deleteidParam != '0')
  (: Удаление данных :)
  then
  (
    local:DeleteData($deleteidParam, $DataListURI)
  )
```

Если был передан HTTP-параметр «deleteid=id удаляемого компьютера», то вызывается функция удаления данных.

```
else if($idParam != '0')
  (: Чтение данных для редактирования в форме :)
  then
  (
    let
      $DataPath :=
concat(xrx_example:PathURI_data_list_in_db($module_id),
xrx_example:MakeUniqueFileNameExtXml($idParam)),
      $FileData := doc($DataPath)//computer
    return $FileData
  )
```

Если был передан HTTP-параметр «id=id компьютера», то возвращаются данные о соответствующем компьютере.

```
else ()
(: ++++++ :)
```

### Локальные функции:

```
(: ++++++ :)  
(: Создание нового элемента данных :)  
(: ++++++ :)  
declare function local:NewData($idParam as xs:string) as node()  
{  
  let $FirstUpgradeId := xrx_example:MakeUniqueName()  
  return  
  <computer id="{ $idParam }">  
    <computer_name/>  
    <processor_id/>  
    <op/>  
    <hdd/>  
    <server>false</server>  
    <upgrades>  
      <upgrade/>  
    </upgrades>  
  </computer>  
};
```

Возвращается новый XML-элемент, соответствующий компьютеру.

```
(: ++++++ :)  
(: Создание нового элемента данных для модернизации :)  
(: ++++++ :)  
declare function local:NewUpgradeData() as node()  
{  
  <upgrades>  
    <upgrade id="">  
      <upgrade_date/>  
      <upgrade_description/>  
    </upgrade>  
  </upgrades>  
};
```

Возвращается новый XML-элемент, соответствующий модернизации компьютера.

```
(: ++++++ :)  
(: Сообщения об ошибках :)  
(: ++++++ :)  
declare function local:CheckErrors($Data as node()) as node()*  
{  
  (  
    if(($Data//op mod 2) != 0)  
      then (<p>Объем ОП должен быть четным числом</p>)  
      else ()  
  ), (  
    if(($Data//hdd mod 2) != 0)  
      then (<p>Объем жесткого диска должен быть четным числом</p>)  
      else ()  
  )  
};
```

Функция осуществляет поиск ошибок в заполненных XML-данных формы.

В качестве параметра функция получает XML-элемент «computer», содержащий заполненные данные формы.

Функция возвращает последовательность элементов «p», содержащих сообщение об ошибках. Если ошибок нет, то последовательность пустая.

В нашем примере осуществляется проверка элементов на четность. Это учебный пример, так как проверить четность элементов можно с помощью XForms.

Здесь хотелось показать, как может быть в принципе организована дополнительная серверная проверка данных.

```
(: ++++++ :)  
(: Сохранение данных :)  
(: ++++++ :)
```

```

declare function local:SaveData($DataListURI as xs:string) as
node()*
{
let
  (: В сессионной переменной хранится строка, поэтому необходимо
  сделать преобразование к элементу :)
  $TempData := element temp {session:get-attribute($current_data)},

```

Данные читаются из сессионной переменной.

Функция `session:get-attribute` возвращает значение сессионной переменной, параметром функции является название переменной.

Поскольку функция `session:get-attribute` возвращает значение сессионной переменной в виде строки, то для преобразования в XML-элемент используется конструкция `element`.

```

  $Data := $TempData//computer,
  $id := xs:string($Data/@id),
  $Errors := local:CheckErrors($Data)

```

Поиск ошибок с помощью приведенной выше функции `local:CheckErrors` и сохранение в переменную `$Errors`.

```

return
  if(count($Errors) = 0)
  (: Сохранение данных :)
  then
  (
    xrx_example:SaveFileInCollection(xrx_example:PathURI_data_list_in_db
    ($module_id), $id, $Data),
    xrx_example:redirect($DataListURI)
  )

```

Если ошибок нет, то данные о компьютере сохраняются в виде отдельного файла XML в коллекции «computers», с помощью функции `xrx_example:SaveFileInCollection`.

После сохранения данных с помощью функции `xrx_example:redirect()` осуществляется перенаправление на XQuery-сценарий, реализующий вывод списка компьютеров.

```

)
(: Сообщения об ошибках :)
else
(
  xrx_example:ShowErrorReturnForm($module_id, $module_ver, "Ошибка
при сохранении данных", $Errors)
)

```

В переменную \$Errors сохраняются сообщения об ошибках. Если количество ошибок больше нуля, то выводится сообщение об ошибках с помощью функции `xrx_example:ShowErrorReturnForm()`.

```
};
```

```

(: ++++++ : )
(: Предварительное сохранение данных :)
(: ++++++ : )
declare function local:PreSaveData() as node() *
{

```

В режиме предварительного сохранения данные читаются из HTTP-запроса и сохраняются в сессионной переменной.

```
session:create(),
```

Если сессия не создана, то она создается.

```
session:set-attribute($current_data, ""),
```

Очистка сессионной переменной. Переменная сценария \$current\_data хранит название сессионной переменной.

```
let
```

```
$PostData := request:get-data(),
```

Чтение данных из HTTP-запроса.

```
$Data := fn:root($PostData)//computer
```

Функция `fn:root` возвращает корневой элемент XML-данных, далее поиск элемента `computer`.

```
return
```

```
session:set-attribute($current_data, $Data)
```

Сохранение в сессионной переменной.

```
};
```

```
(: ++++++ :)
```

```
(: Удаление данных :)
```

```
(: ++++++ :)
```

```
declare function local:DeleteData($deleteidParam as xs:string,  
$DataListURI as xs:string) as node()*  
{
```

```
xrx_example:DeleteFileInCollection(xrx_example:PathURI_data_list_in_  
db($module_id), $deleteidParam),  
  xrx_example:redirect($DataListURI)
```

При удалении данных о компьютере происходит удаление файла из соответствующей коллекции БД.

```
};
```

#### **7.3.5.5.4 Взаимодействие между сценариями справочника «Компьютер»**



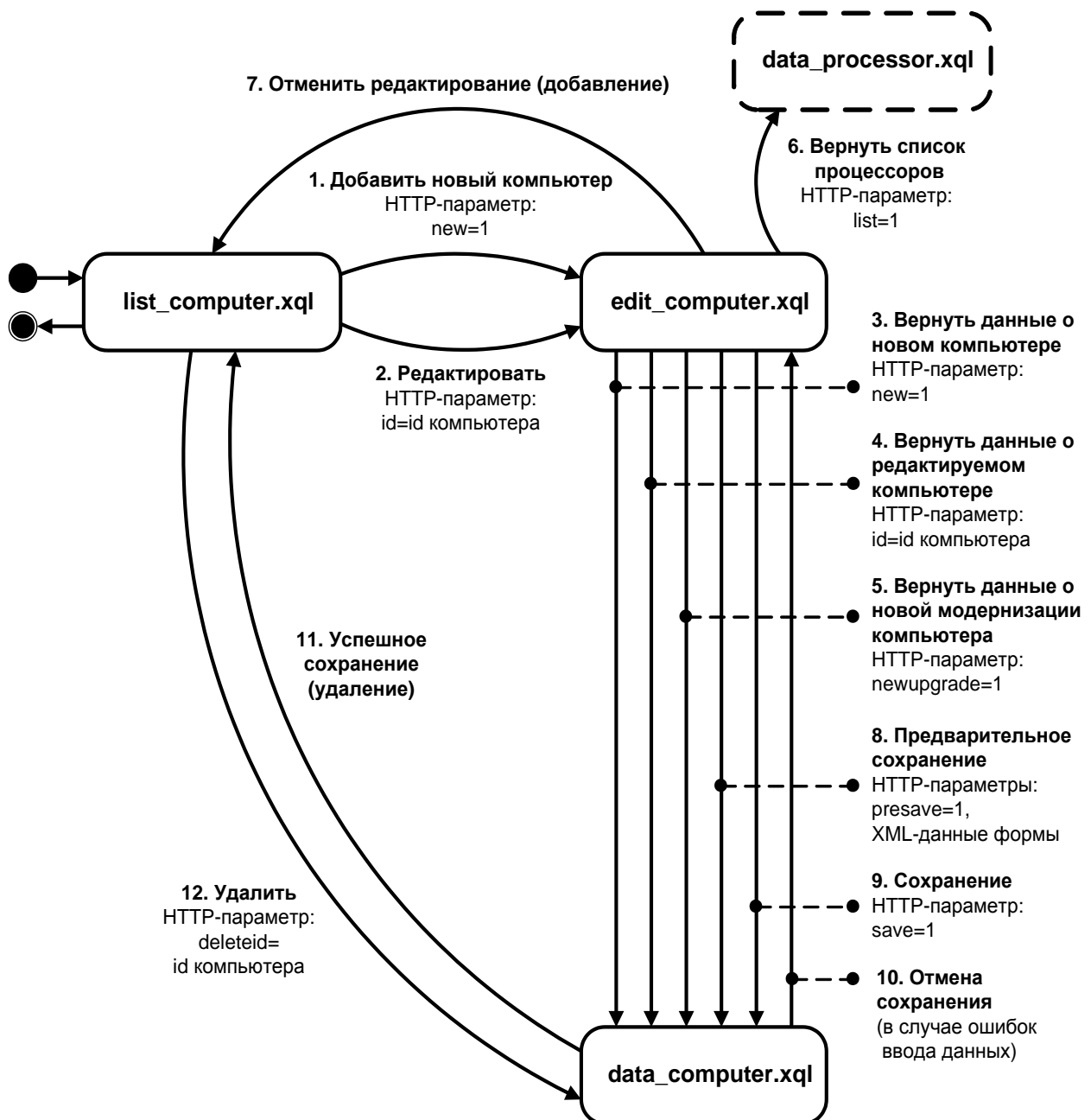


Рис. 7.11. Взаимодействие между сценариями справочника «Компьютер».

Переход из начального состояния диаграммы осуществляется на сценарий «list\_computer.xql», так как пользователь начинает работу со справочником компьютеров со списка данных.

Также из этого сценария осуществляется переход на конечное состояние. Этому переходу соответствует кнопка «Вернуться в главное меню».

В случае нажатия кнопки добавления нового элемента (1) осуществляется вызов сценария редактирования элемента «edit\_computer.xml». При этом передается HTTP-параметр «new=1».

В случае нажатия кнопки редактирования элемента (2) также осуществляется вызов сценария редактирования элемента «edit\_computer.xml». При этом передается HTTP-параметр «id=id редактируемого компьютера».

Получив этот параметр, сценарий «edit\_computer.xml» генерирует XForms-форму ввода (редактирования) данных. В обоих случаях XForms-форма обращается к сценарию «data\_computer.xml» для чтения данных.

В случае добавления нового компьютера (3) сценарию «data\_computer.xml» передается HTTP-параметр «new=1», и сценарий возвращает незаполненный XML-элемент для ввода нового компьютера.

В случае редактирования данных о компьютере (4) сценарию «data\_computer.xml» передается HTTP-параметр «id=id компьютера», и сценарий возвращает данные о компьютере.

Для получения незаполненного элемента «upgrade», который используется при добавлении данных о модернизации компьютера, (5) XForms-форма вызывает сценарий «data\_computer.xml» с HTTP-параметром «newupgrade=1».

Для получения списка процессоров (6) XForms-форма обращается к модулю «тип процессора», вызывая сценарий «data\_processor.xml» с HTTP-параметром «list=1».

В случае нажатия в форме редактирования (добавления) кнопки «Отменить» (7) осуществляется возврат в список данных.

В случае нажатия в форме редактирования (добавления) кнопки «Сохранить» сначала происходит асинхронная передача заполненной формы на сервер и предварительное сохранение данных в переменной сессии (8). Затем происходит синхронный вызов сценария (9), при котором данные читаются из сессионной переменной, проверяются и сохраняются в БД.

Если данные введены неправильно, то выводится сообщение об ошибке и происходит возврат в форму редактирования (10).

Если данные введены правильно, то после сохранения данных в БД осуществляется переход в форму списка (11).

Если в форме списка «list\_computer.xml» нажата кнопка удаления данных (12), то осуществляется вызов сценария «data\_computer.xml». Ему передается параметр HTTP-запроса «deleteid=id удаляемого компьютера». После удаления данных осуществляется переход в форму списка (11).

### 7.3.5.6 Модуль формирования отчета

Модуль формирования отчета используется для ввода параметров отчета, формирования запроса к БД на основе параметров, получения данных и вывода данных в виде HTML.

The screenshot shows a Mozilla Firefox browser window with the address bar displaying `http://localhost:8080/exist/xrx_example/report.xml`. The page title is "Параметры отчета". The form contains the following fields and controls:

- Подстрока, которую должно содержать наименование компьютера:** A text input field.
- Типы процессоров:** A dropdown menu with options: Core 2 Duo, Pentium, and Pentium IV.
- Ограничение для объема ОП:** Two input fields labeled "Минимальное:" and "Максимальное:".
- Используется как сервер:** A dropdown menu with the option "Нет фильтра".
- Были модернизации в диапазоне:** Two input fields labeled "От:" and "До:".
- Сортировка по полю:** Two dropdown menus with options "Название компьютера" and "по возрастанию".
- Форматировать отчет:** A dropdown menu with the option "таблицей".
- Сформировать отчет:** A button.

The status bar at the bottom of the browser window shows "Готово".

Рис. 7.12. Ввод параметров отчета.

**Результат отчета**

Наименование	Тип процессора	Объем ОП (в Мб)	Объем жесткого диска (в Гб)	Используется как сервер	Обновления						
Сервер 1	Core 2 Duo	1024	1000	Да	<table border="1"> <thead> <tr> <th>Дата</th> <th>Описание</th> </tr> </thead> <tbody> <tr> <td>2010-10-04</td> <td>увеличение ОП</td> </tr> <tr> <td>2010-10-05</td> <td>замена диска</td> </tr> </tbody> </table>	Дата	Описание	2010-10-04	увеличение ОП	2010-10-05	замена диска
Дата	Описание										
2010-10-04	увеличение ОП										
2010-10-05	замена диска										

**Текст запроса**

```
(: report_data - корневой элемент данных :)
<report_data>
{
  let
    (: Документ, содержащий информацию о процессорах :)
    $Processors := doc('/db/iu5/xrx_example/processors/processors.xml')
    (: Копирование содержимого данных о компьютерах :)

```

Готово

Рис. 7.13. Сформированный отчет.

В модуле формирования отчета также используется вариант архитектуры II, (технология XForms). Однако особенностью этого модуля является то, что все необходимые действия выполняет единственный XQuery-сценарий.

Для реализации модуля необходимо разработать:

- report.xql – серверный XQuery-сценарий.
- report.xsl – XSLT-преобразование, которое используется для форматирования результирующих данных.

### 7.3.5.6.1 Сценарий report.xql

Серверный сценарий для формирования отчета.

Пролог сценария:

```
xquery version "1.0";

(: Импорт стандартных модулей :)
declare namespace request="http://exist-db.org/xquery/request";
declare namespace session="http://exist-db.org/xquery/session";
(: Импорт модулей приложения :)
import module namespace xrx_example =
"http://iu5.bmstu.ru/edu/xrx_example" at "module.xqm";
(: Формат, в котором результат должен отображаться в браузере :)
declare option exist:serialize "method=xhtml media-type=text/html
indent=no";
(: Идентификатор модуля :)
declare variable $module_id {"report"};
(: Версия модуля :)
declare variable $module_ver {" "};
(: Для вызова модуля процессора :)
declare variable $module_processor_id {"processor"};
declare variable $module_processor_ver {"1.2"};
(: Для вызова модуля компьютера :)
declare variable $module_computer_id {"computer"};
declare variable $module_computer_ver {"1.1"};
(: Название сессионной переменной для хранения данных с параметрами
отчета :)
declare variable $report_params_data {"report_params_data"};
```

Основной запрос сценария:

```
(: ++++++ :)
```

```
(: Основной запрос :)
```

```
(: ++++++ :)
```

```
(: Если не заполнен справочник типов процессоров, то формирование
отчета невозможно :)
```

```

if (xrx_example:ModuleDataCount($module_processor_id,
$module_processor_ver) = 0)
then
  (: Вывод сообщения об ошибке :)
  (
    xrx_example:ShowErrorButton("Невозможно формирование отчета, так
как не заполнен справочник типов процессоров", "", "")
  )
  (: Если не заполнен справочник компьютеров, то формирование отчета
невозможно :)
else if (xrx_example:ModuleDataCount($module_computer_id,
$module_computer_ver) = 0)
then
  (: Вывод сообщения об ошибке :)
  (
    xrx_example:ShowErrorButton("Невозможно формирование отчета, так
как не заполнен справочник компьютеров", "", "")
  )

```

**Формирование отчета возможно, только если заполнены справочники типов процессоров и компьютеров.**

```

else
  (: Выполнение основных действий :)
  (
let
  $reportParam := xs:string(request:get-parameter("report","0")),
  $prereportParam := xs:string(request:get-
parameter("prereport","0")),
  $DataListURI := xrx_example:PathURI_list($module_id, $module_ver)

```

**Чтение HTTP-параметров и сохранение во временные переменные.**

```

return
  if($reportParam = '1')
    (: Формирование отчета :)
    then
      (
        local:CheckErrorsAndMakeReport($DataListURI)

```

[Оглавление](#)

)

Если был передан HTTP-параметр «report=1», то производится чтение предварительно сохраненных параметров из сессионной переменной, выполняется проверка корректности ввода параметров. Если параметры введены корректно, то формируется отчет.

```
else if($prereportParam = '1')
  (: Сохранение условий для формирования отчета :)
  then
  (
    local:PreReport()
  )
```

Если был передан HTTP-параметр «prereport=1», то заполненные XML-данные из формы параметров отчета предварительно сохраняются в сессионной переменной.

```
else
  (: Генерация формы ввода данных :)
  (
    local:ReportForm()
  )
```

Если сценарию не были переданы HTTP-параметры, то по умолчанию сценарий генерирует XForms-форму ввода параметров отчета.

```
)
(: ++++++ :)
```

```
(: ++++++ :)
```

```
(: Сообщения об ошибках при вводе параметров отчета :)
```

```
(: ++++++ :)
```

```
declare function local:CheckReportErrors($Data as node()) as node()*
{
```

Функция осуществляет поиск ошибок в заполненных XML-данных формы параметров отчета.

В качестве параметра функция получает XML-элемент «report\_params», содержащий заполненные данные формы.

Функция возвращает последовательность элементов «р», содержащих сообщение об ошибках. Если ошибок нет, то последовательность пустая.

```
(
(: Проверка диапазона имеет смысл, только если заданы оба значения
:))
if(($Data//op1 castable as xs:integer) and ($Data//op2 castable as
xs:integer))
```

Проверка для полей «минимальное и максимальное ограничение для объема ОП». Если оба элемента «ор1» и «ор2» можно привести к целому типу, то осуществляется проверка, что минимальное значение диапазона («ор1») меньше или равно максимальному значению диапазона («ор2»).

Далее такая же проверка производится для минимального и максимального значения поля «Были модернизации в диапазоне».

```
then
(
let
$op1 := xs:integer($Data//op1),
$op2 := xs:integer($Data//op2)
return
if($op1 > $op2)
then (<p>Минимальное ограничение для объема ОП должно быть
меньше или равно максимальному</p>)
else ()
)
else ()
), (
(: Проверка диапазона имеет смысл, только если заданы оба значения
:))
if(($Data//upgrade_date1 castable as xs:date) and
($Data//upgrade_date2 castable as xs:date))
then
(
let
$upgrade_date1 := xs:date($Data//upgrade_date1),
```



```

    $upgrade_date2 := xs:date($Data//upgrade_date2)
return
    if($upgrade_date1 > $upgrade_date2)
        then (<p>Минимальное ограничение для даты модернизации
должно быть меньше или равно максимальному</p>)
        else ()
    )
else ()
)
};

(: ++++++ : )
(: Проверка ошибок и формирование отчета :)
(: ++++++ : )
declare function local:CheckErrorsAndMakeReport($DataListURI as
xs:string) as node()*
{
let
    (: В сессионной переменной хранится строка, поэтому необходимо
сделать преобразование к элементу :)
    $TempData := element temp {session:get-
attribute($report_params_data)},
    $Data := $TempData//report_params,

```

Данные читаются из сессионной переменной.

Функция session:get-attribute возвращает значение сессионной переменной, параметром функции является название переменной.

Поскольку функция session:get-attribute возвращает значение сессионной переменной в виде строки, то для преобразования в XML-элемент используется конструкция element.

```
$Errors := local:CheckReportErrors($Data)
```

Поиск ошибок с помощью приведенной выше функции local:CheckReportErrors и сохранение в переменную \$Errors.

```

return
    if(count($Errors) = 0)

```

```

(: Формирование отчета :)
then
(
  local:Report($Data)
  Если нет ошибок, то производится формирование отчета.
)
(: Сообщения об ошибках :)
else
(
  xrx_example:ShowErrorReturnForm("", "", "Ошибка при вводе
параметров отчета", $Errors)
  В переменную $Errors сохраняются сообщения об ошибках. Если количество
ошибок больше нуля, то выводится сообщение об ошибках с помощью функции
xrx_example:ShowErrorReturnForm().
)
};

```

```

(: ++++++ : )
(: Преобразование XSLT для отчета :)
(: ++++++ : )
declare function local:transform_report($param as node(),
$report_format as xs:string) as node()*
{

```

Функция выполняет XSLT-преобразование данных, которые были получены в результате выполнения динамически сформированного запроса в HTML.

```

(: Параметры преобразования :)
let
  $XSLTFile := concat(xrx_example:BaseCurrentURL(), "report.xsl"),
  Для преобразования используется файл "report.xsl".
  $params :=
    <parameters>
      <param name="report_format" value="{ $report_format }"/>
    </parameters>

```

В качестве параметра в преобразование передается тип форматирования отчета (списком или таблицей), этот параметр задается в форме ввода параметров.

```
let $TransformRes := transform:transform($param,
xs:anyURI($XSLTFile), $params)
return $TransformRes
```

Функция возвращает HTML-фрагмент, который соответствует результирующим данным, представленным в виде таблицы или списка.

```
};
```

```
(: ++++++ :)
```

```
(: Формирование запроса для отчета.
```

Запрос формируется в виде строки, содержащей XQuery-запрос.

В дальнейшем этот запрос будет динамически интерпретирован и выполнен с помощью функции `util:eval`

```
(аналогично динамическому SQL) :)
```

```
(: ++++++ :)
```

```
declare function local:MakeReportQuery($Data as node()) as xs:string
{
```

(Размер шрифта здесь немного уменьшен, чтобы сохранилось форматирование.)

В этой функции динамически формируется XQuery-запрос на выборку данных на основе параметров отчета, введенных в форме. Задача функции – сформировать строку, содержащую XQuery-запрос.

Этот запрос снабжен большим количеством комментариев, поэтому здесь мы только поясним подход к формированию таких запросов.

Вначале выбираются необходимые для отчета данные с помощью вложенных конструкций `for` и `let`.

Затем формируется «пустой» оператор `where` (`1=1`).

Далее проверяется каждый параметр отчета. Если параметр заполнен, то соответствующее этому параметру условие добавляется в секцию «where».

Например, если задан фрагмент наименования компьютера, то в отчет попадают только такие компьютеры, в название которых входит этот фрагмент.

```
let
```

```

$t1 := concat("(: report_data - корневой элемент данных :)", "&#13;",
    "<report_data>", "&#13;", "{ ", "&#13;",
    " let ", "&#13;",
    " (: Документ, содержащий информацию о процессорах :)", "&#13;",
    " $Processors := doc(' ', xrx_example:ProcessorsFileName(), '')//processors, ", "&#13;",
    " (: Коллекция, содержащая данные о компьютерах :)", "&#13;",
    " $ComputersList := collection(' ', xrx_example:PathURI_data_list_in_db('computer'), '')
", "&#13;",
    " for ", "&#13;",
    " (: Перебор всех всех файлов в коллекции, из каждого файла читается корневой элемент
computer :)", "&#13;",
    " $Computer in $ComputersList//computer ", "&#13;",
    " let ", "&#13;",
    " (: Название компьютера выделяется в отдельную переменную для удобства сортировки :)",
"&#13;",
    " $computer_name := $Computer/computer_name, ", "&#13;",
    " (: Для каждого компьютера по id процессора читается название процессора :)", "&#13;",
    " $processor_name := $Processors//processor[@id=$Computer/processor_id]/processor_name ",
"&#13;",
    " (: where - условия выборки, (1=1) - заглушка для where, если далее не задано никаких
условий :)", "&#13;",
    " where (1=1) ", "&#13;"),

    $t2 := concat($t1,
        if($Data/computer_name = "") then ("")
        else
        (
            concat(
                " (: Если пользователь указал часть названия компьютера, то этот параметр должен входить как
подстрока в названия компьютеров :)", "&#13;",
                " and (fn:contains($Computer/computer_name, ' ', $Data/computer_name, '')) ", "&#13;"
            )
        )),

    $t3 := concat($t2,
        if($Data/processor_id = "") then ("")
        else
        (
            concat(
                " (: Параметр отчета 'процессор' возвращается в виде набора id-элементов, соединенных
пробелами :)", "&#13;",
                " (: Поэтому нужно проверить, не входит ли id текущего компьютера в эту строку :)", "&#13;",
                " and (fn:contains(' ', $Data/processor_id, ' ', $Computer/processor_id)) ", "&#13;"
            )
        )),

    $t4 := concat($t3,
        if($Data/op1 castable as xs:integer)
        then
        (
            concat(
                " (: Ограничение снизу на объем ОП :)", "&#13;",
                " and (xs:integer($Computer/op) >= xs:integer(' ', $Data/op1, '')) ", "&#13;"
            )
        )
    )

```

```

    )
  )
  else ("")),

$t5 := concat($t4,
  if($Data/op2 castable as xs:integer)
  then
  (
    concat(
      "    (: Ограничение сверху на объем ОП :)", "&#13;",
      "    and (xs:integer($Computer/op) <= xs:integer('", $Data/op2, "')) ", "&#13;"
    )
  )
  else ("")),

$t6 := concat($t5,
  if($Data/server = "nofilter") then ("")
  else
  (
    concat(
      "    (: Проверка признака сервера :)", "&#13;",
      "    and ($Computer/server = '", $Data/server, "') ", "&#13;"
    )
  )),

(: Проверка того, что в заданном диапазоне было хотя бы одно обновление :)
(: Проверка что существует хотя бы один элемент upgrade, дата которого соответственно больше или
меньше заданной границы :)
(: Здесь могла быть более сложная проверка на пересечение диапазона в запросе и диапазона обновлений
:))

$t7 := concat($t6,
  if($Data/upgrade_date1 castable as xs:date)
  then
  (
    concat(
      "    (: Проверка того, что хотя бы одно обновление было позже параметра :)", "&#13;",
      "    and (fn:exists($Computer/upgrades/upgrade[xs:date(upgrade_date) >= xs:date('",
$Data/upgrade_date1, "']))) ", "&#13;"
    )
  )
  else ("")),
$t8 := concat($t7,
  if($Data/upgrade_date2 castable as xs:date)
  then
  (
    concat(
      "    (: Проверка того, что хотя бы одно обновление было ранее параметра :)", "&#13;",
      "    and (fn:exists($Computer/upgrades/upgrade[xs:date(upgrade_date) <= xs:date('",
$Data/upgrade_date2, "']))) ", "&#13;"
    )
  )
  else ("")),

```

```

$OrderBy := concat($t8,
    "(: Сортировка :)", "&#13;",
    "order by ", $Data/order_field, " ", $Data/order_type, " ", "&#13;"
),

$Result := concat($OrderBy,
    "(: Возвращаемые данные :)", "&#13;",
    "return ", "&#13;",
    " <computer> ", "&#13;", " { ", "&#13;",
    "   $Computer/computer_name, ", "&#13;",
    "   $processor_name, ", "&#13;",
    "   $Computer/op, ", "&#13;",
    "   $Computer/hdd, ", "&#13;",
    "   $Computer/server, ", "&#13;",
    "   $Computer/upgrades ", "&#13;",
    " }", "&#13;", " </computer> ", "&#13;",
    "}", "&#13;", "</report_data> ")

return $Result
};

```

```

(: ++++++ : )
(: Формирование отчета :)
(: ++++++ : )
declare function local:Report($Data as node()) as node() *
{
let
    $ReportFormat := xs:string($Data/report_format),
        Как форматировать отчет – списком или таблицей.
    $CurrentCssURI := xrx_example:PathURI_current_css(),
        Путь к таблице стилей CSS.
    $Query := local:MakeReportQuery($Data),
        Формирование текста XQuery-запроса на выборку данных.
    $ReportData := util:eval($Query),
        Выполнение запроса и получение данных.
    $XSLTResult := local:transform_report($ReportData, $ReportFormat)
        Преобразование данных в HTML с помощью XSLT-преобразования.
return
    Формирование результирующего документа в формате HTML.

```

```

<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8"/>
    <style type="text/css">
      <!-- Добавление стилей CSS с помощью XInclude -->
      <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
href="{ $CurrentCssURI }"/>
    </style>
  </head>
  <body>
    <h1>Результат отчета</h1>
    <div>{$XSLTResult}</div>
    Переменная $XSLTResult содержит данные отчета, которые были
преобразованы в HTML с помощью XSLT-преобразования.
    <hr/>
    <h2>Текст запроса</h2>
    <big><big><pre>{$Query}</pre></big></big>
    Переменная $Query содержит текст запроса и выводится для справки.
  </body>
</html>
};

```

```

(: ++++++ : )
(: Сохранение условий для формирования отчета :)
(: ++++++ : )
declare function local:PreReport() as node() *

```

Параметры отчета, переданные из XForms-формы, сохраняются в сессионную переменную.

```

{
  (: Если сессия не создана, то она создается :)
  session:create(),

```

```

session:set-attribute($report_params_data, ""),
let
    $PostData := request:get-data(),
    $Data := fn:root($PostData)//report_params
return
    session:set-attribute($report_params_data, $Data)
};

(: ++++++ : )
(: Модели данных формы :)
(: ++++++ : )
declare function local:CreateXFormsModels($ProcessorReadURI as
xs:string, $PreReportURI as xs:string) as node()*
{

```

**Функция генерирует модель данных для формы ввода параметров отчета и ограничения на вводимые параметры с помощью xforms:bind.**

```

(
(: Основная модель данных :)
<xforms:model id="computer_model">

    <!-- Основные данные (о компьютере) -->
    <xforms:instance id="computer_instance" xmlns="">
        <report_params>
            <computer_name/>
            <processor_id/>
            <op1/>
            <op2/>
            <server>nofilter</server>
            <upgrade_date1/>
            <upgrade_date2/>
            <order_field>$computer_name</order_field>
            <order_type>ascending</order_type>
            <report_format>table</report_format>
        </report_params>
    </xforms:instance>

```





```

        <xforms:input model="computer_model" ref="computer_name"/>
    </td>
</tr>

<tr>
    <th colspan="2" align="right">Типы процессоров:</th>
    <td>
        <xforms:select model="computer_model" ref="processor_id">
            <xforms:itemset model="processor_model"
nodeset="//processor">
                <xforms:value ref="@id"/>
                <xforms:label ref="processor_name"/>
            </xforms:itemset>
        </xforms:select>
    </td>
</tr>

<tr>
    <th rowspan="2" align="right">Ограничение для объема ОП</th>
    <th align="right">Минимальное:</th>
    <td>
        <xforms:input model="computer_model" ref="op1">
            <xforms:alert>Значение должно быть целым
числом</xforms:alert>
        </xforms:input>
    </td>
</tr>

<tr>
    <th align="right">Максимальное:</th>
    <td>
        <xforms:input model="computer_model" ref="op2">
            <xforms:alert>Значение должно быть целым
числом</xforms:alert>
        </xforms:input>
    </td>
</tr>

```

```

        </td>
    </tr>

    <tr>
        <th colspan="2" align="right">Используется как сервер:</th>
        <td>
            <xforms:select1 model="computer_model" ref="server">
                <xforms:choices>
                    <xforms:item>
                        <xforms:label>Нет фильтра</xforms:label>
                        <xforms:value>nofilter</xforms:value>
                    </xforms:item>
                    <xforms:item>
                        <xforms:label>Да</xforms:label>
                        <xforms:value>true</xforms:value>
                    </xforms:item>
                    <xforms:item>
                        <xforms:label>Нет</xforms:label>
                        <xforms:value>>false</xforms:value>
                    </xforms:item>
                </xforms:choices>
            </xforms:select1>
        </td>
    </tr>

    <tr>
        <th rowspan="2" align="right">Были модернизации в
диапазоне</th>
        <th align="right">От:</th>
        <td>
            <xforms:input model="computer_model" ref="upgrade_date1">
                <xforms:alert>Значение должно быть датой</xforms:alert>
            </xforms:input>
        </td>
    </tr>

```

```

<tr>
  <th align="right">До:</th>
  <td>
    <xforms:input model="computer_model" ref="upgrade_date2">
      <xforms:alert>Значение должно быть датой</xforms:alert>
    </xforms:input>
  </td>
</tr>

<tr>
  <th colspan="3" align="center">Сортировка по полю:

  <xforms:select1 model="computer_model" ref="order_field">
    <xforms:choices>
      <xforms:item>
        <xforms:label>Название компьютера</xforms:label>
        <xforms:value>$computer_name</xforms:value>
      </xforms:item>
      <xforms:item>
        <xforms:label>Название процессора</xforms:label>
        <xforms:value>$processor_name</xforms:value>
      </xforms:item>
    </xforms:choices>
  </xforms:select1>

  <xforms:select1 model="computer_model" ref="order_type">
    <xforms:choices>
      <xforms:item>
        <xforms:label>по возрастанию</xforms:label>
        <xforms:value>ascending</xforms:value>
      </xforms:item>
      <xforms:item>
        <xforms:label>по убыванию</xforms:label>
        <xforms:value>descending</xforms:value>

```

```

        </xforms:item>
    </xforms:choices>
</xforms:select1>
    </th>

</tr>

<tr>
    <th colspan="2" align="right">Форматировать отчет:</th>
    <td>
        <xforms:select1 model="computer_model" ref="report_format">
            <xforms:choices>
                <xforms:item>
                    <xforms:label>таблицей</xforms:label>
                    <xforms:value>table</xforms:value>
                </xforms:item>
                <xforms:item>
                    <xforms:label>списком</xforms:label>
                    <xforms:value>list</xforms:value>
                </xforms:item>
            </xforms:choices>
        </xforms:select1>
    </td>
</tr>

<tr>
    <td colspan="3" align="center">

        <xforms:trigger>
            <xforms:label>Сформировать отчет</xforms:label>
            <xforms:action ev:event="DOMActivate" if="is-
valid(instance('computer_instance'))">
                <!-- Данные передаются на сервер и сохраняются в сессионной
переменной -->
                <xforms:send submission="submit_id" />

```

```
<!-- Данные читаются из сессионной переменной, проверяются и
передаются для формирования отчета -->
```

```
<xforms:load show="replace" resource="{ $ReportURI }"/>
```

```
</xforms:action>
```

```
</xforms:trigger>
```

```
</td>
```

```
</tr>
```

```
</table>
```

```
</div>
```

```
};
```

```
(: ++++++ :)
```

```
(: Форма параметров отчета :)
```

```
(: ++++++ :)
```

```
declare function local:ReportForm() as node()
```

```
{
```

**Функция генерирует XHTML-документ, содержащий XForms-форму ввода параметров отчета.**

```
let
```

```
$ProcessorReadURI :=
```

```
xrx_example:PathURI_data_list($module_processor_id,
```

```
$module_processor_ver),
```

```
(: К названию текущего скрипта добавляется параметр ?prereport=1 :)
```

```
$CurrentScriptName := xrx_example:FileNameFromCurrentURL(),
```

```
$PreReportURI := concat($CurrentScriptName, "?prereport=1"),
```

```
$ReportURI := concat($CurrentScriptName, "?report=1"),
```

```
$CurrentCssURI := xrx_example:PathURI_current_css(),
```

```
$XFormsCssURI := xrx_example:PathURI_xforms_css(),
```

```
$form :=
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
xmlns:xforms="http://www.w3.org/2002/xforms"
```

```
xmlns:ev="http://www.w3.org/2001/xml-events">
```

```
<head>
```

```

<meta http-equiv="Content-Type" content="text/html; charset=UTF-
8"/>
<style type="text/css">
    <!-- Добавление стилей CSS с помощью XInclude -->
    <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
href="{ $CurrentCssURI }"/>
    <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
href="{ $XFormsCssURI }"/>
</style>
<!-- Модель создается в секции head -->
{ local:CreateXFormsModels($ProcessorReadURI, $PreReportURI) }
</head>
<body>
    <h1>Параметры отчета</h1>
    <!-- Поля редактирования формы -->
    { local:EditForm($ReportURI) }
</body>
</html>
(: сгенерированный XForms-документ обрабатывается с помощью XSLT-
преобразования для получения HTML-документа :)
return xrx_example:transform_xsltforms($form)
};

```

#### 7.3.5.6.2 XSLT-преобразование report.xsl

Файл «report.xsl» используется для преобразования данных отчета из XML в HTML.

На вход преобразование получает данные отчета (корневой элемент «report\_data»), на выходе генерируется фрагмент HTML-документа.

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:param name="report_format" select="string('table')"/>

```

Параметр «report\_format», который передается в преобразование, определяет тип форматирования отчета – в виде таблицы или в виде списка.

```

<!-- ++++++ -->
<!-- Шаблон обработки корневого элемента XML - документа -->
<xsl:template match="/">
  <div>
    <xsl:apply-templates/>
  </div>
</xsl:template>

```

Шаблон для корневого элемента оборачивает результат преобразования в элемент «div».

```

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

<!-- ++++++ -->
<xsl:template match="report_data">

```

Шаблон преобразования для основного элемента данных отчета «report\_data».

```

<xsl:choose>
  <xsl:when test="$report_format='table'">

```

Если задано форматирование в виде таблицы (\$report\_format='table'), то формируется шапка таблицы, затем данные выводятся в виде строк таблицы.

```

  <table border="1">
    <tr>
      <th>Наименование</th>
      <th>Тип процессора</th>
      <th>Объем ОП (в Мб)</th>
      <th>Объем жесткого диска (в Гб)</th>
      <th>Используется как сервер</th>
      <th>Обновления</th>
    </tr>

```

```

  <xsl:for-each select="computer">
    <tr>
      <td><xsl:value-of select="computer_name"/></td>

```



```

<td><xsl:value-of select="processor_name"/></td>
<td><xsl:value-of select="op"/></td>
<td><xsl:value-of select="hdd"/></td>
<td>
  <xsl:choose>
    <xsl:when test="server[.='true']">Да</xsl:when>
    <xsl:otherwise>Нет</xsl:otherwise>
  </xsl:choose>
</td>
<td><xsl:apply-templates select="upgrades"/></td>
</tr>
</xsl:for-each>
</table>
</xsl:when>

```

```
<xsl:otherwise>
```

Иначе, если задано форматирование в виде списка, то данные выводятся в виде элементов нумерованного списка.

```

<ol>
  <xsl:for-each select="computer">
    <li>
      <p>Наименование: <b><xsl:value-of
select="computer_name"/></b></p>
      <p>Тип процессора: <b><xsl:value-of
select="processor_name"/></b></p>
      <p>Объем ОП (в Мб): <b><xsl:value-of
select="op"/></b></p>
      <p>Объем жесткого диска (в Гб): <b><xsl:value-of
select="hdd"/></b></p>
      <p>Используется как сервер: <b>
        <xsl:choose>
          <xsl:when test="server[.='true']">Да</xsl:when>
          <xsl:otherwise>Нет</xsl:otherwise>
        </xsl:choose></b>
      </p>

```

```

        <p><xsl:apply-templates select="upgrades"/></p>
    </li>
</xsl:for-each>
</ol>
</xsl:otherwise>
</xsl:choose>

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

<!-- ++++++ -->
<xsl:template match="upgrades">

```

Модернизации компьютера (элементы «upgrade») также выводятся в виде списка или в виде таблицы.

```

    <xsl:choose>
        <xsl:when test="$report_format='table'">

            <xsl:choose>
                <xsl:when test="count(upgrade[upgrade_date and
upgrade_description]) > 0">

                    <table border="1">
                        <tr>
                            <th>Дата</th>
                            <th>Описание</th>
                        </tr>

                        <xsl:for-each select="upgrade[upgrade_date and
upgrade_description]">
                            <tr>
                                <td><xsl:value-of select="upgrade_date"/></td>
                                <td><xsl:value-of select="upgrade_description"/></td>
                            </tr>
                        </xsl:for-each>
                    </table>

```

```

        </xsl:when>

        <xsl:otherwise>
            <p>Нет обновлений</p>
        </xsl:otherwise>
    </xsl:choose>
</xsl:when>

<xsl:otherwise>
    <xsl:if test="count(upgrade[upgrade_date and
upgrade_description]) > 0">
        <p>Список обновлений:</p>
        <ol>
            <xsl:for-each select="upgrade[upgrade_date and
upgrade_description]">
                <li><b><xsl:value-of select="upgrade_date"/>
(<xsl:value-of select="upgrade_description"/>)</b></li>
            </xsl:for-each>
        </ol>
    </xsl:if>

</xsl:otherwise>
</xsl:choose>

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

</xsl:stylesheet>

```

#### 7.3.5.6.3 Взаимодействие между сценариями модуля

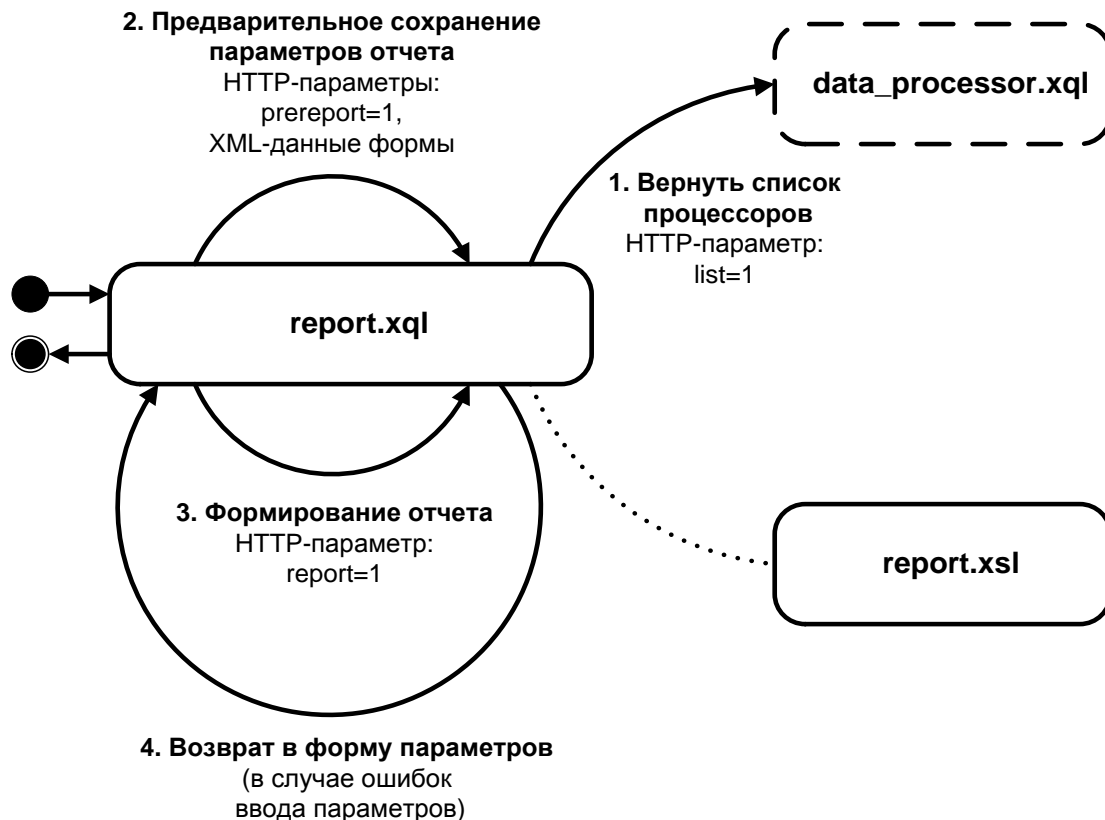


Рис. 7.14. Взаимодействие между сценариями модуля.

Переход из начального состояния диаграммы на сценарий «report.xql» соответствует вызову сценария без параметров. В этом режиме сценарий генерирует XForms-форму ввода параметров отчета.

Для получения списка процессоров (1) XForms-форма обращается к модулю «тип процессора», вызывая сценарий «data\_processor.xql» с HTTP-параметром «list=1».

В случае нажатия в форме кнопки «Сформировать отчет» сначала происходит асинхронная передача заполненной формы параметров отчета в серверный сценарий (в нашем случае это тот же сценарий «report.xql») и предварительное сохранение параметров отчета в переменной сессии (2).

Затем происходит синхронный вызов сценария «report.xql» (3), при котором параметры отчета читаются из сессионной переменной и проверяются.

Если данные введены неправильно, то выводится сообщение об ошибке и происходит возврат в форму редактирования параметров (4).

Если данные введены правильно, то происходит формирование отчета, для преобразования результирующих данных отчета в HTML применяется XSLT-преобразование «report.xsl».

Переход в конечное состояние соответствует закрытию окна браузера, в котором был открыт отчет. Поскольку сценарий формирования отчета открывается в новом окне браузера, то возврат в главное меню здесь не используется.

#### **7.3.5.7 Другие сценарии**

В этом разделе рассмотрим остальные файлы нашего примера, которые не вошли в справочники и формирование отчета:

- index.xql – сценарий, реализующий главное меню.
- initdb.xql – сценарий инициализации БД перед работой с приложением.
- module.xqm – XQuery-модуль, содержащий вспомогательные функции.
- style.css – основная стилевая таблица.
- xforms.css – стилевая таблица для отображения XForms-форм.

##### **7.3.5.7.1 Сценарий index.xql**

Сценарий предназначен для формирования главного меню.

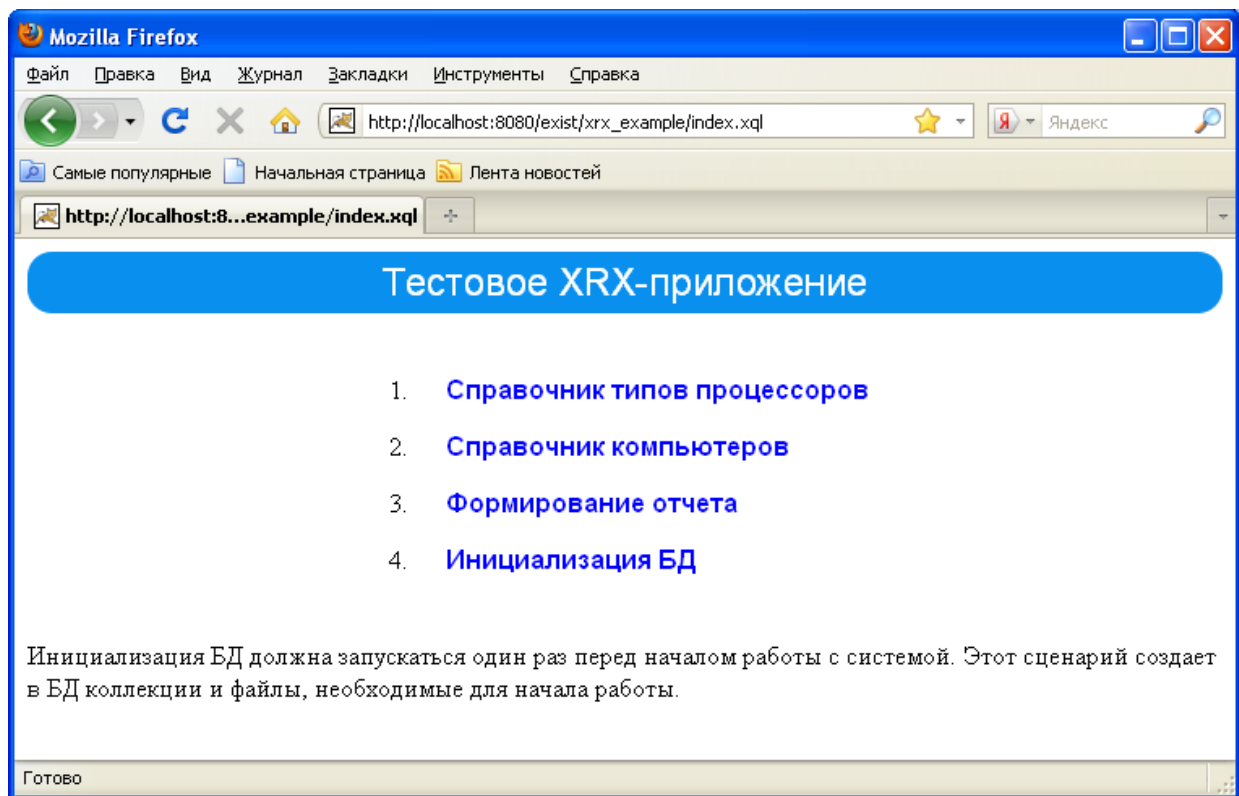


Рис. 7.15. Сценарий index.xql

Текст сценария:

```
xquery version "1.0";

(: Импорт стандартных модулей :)

declare namespace request="http://exist-db.org/xquery/request";
declare namespace session="http://exist-db.org/xquery/session";

(: Импорт модулей приложения :)

import module namespace xrx_example =
"http://iu5.bmstu.ru/edu/xrx_example" at "module.xqm";

(: Формат, в котором результат должен отображаться в браузере :)

declare option exist:serialize "method=xhtml media-type=text/html
indent=no";

(: Идентификатор модуля :)

declare variable $module_id {"main"};

(: Версия модуля :)

declare variable $module_ver {""};

(: ++++++ : )

(: Основной запрос :)

(: ++++++ : )
```

## Оглавление

```

let $CurrentCssURI := xrx_example:PathURI_current_css()
return
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8"/>
    <style type="text/css">
      <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
href="{ $CurrentCssURI }"/>

```

Подсветка пунктов меню реализована с использованием CSS-стилей.

```

  </style>
</head>
<body>
  <h1>Тестовое XRX-приложение</h1>
  <table align="center">
    <tr>
      <td>
        <ol class="menu">
          <li><a class="menu"
href="list_processor_1.2.xql">Справочник типов процессоров</a></li>
          <li><a class="menu"
href="list_computer_1.1.xql">Справочник компьютеров</a></li>
          <li><a class="menu" target="_blank"
href="report.xql">Формирование отчета</a></li>

```

Пункт меню «Формирование отчета» открывается в новом окне (вкладке) браузера, так как указан атрибут `target="_blank"`.

```

          <li><a class="menu" href="initdb.xql">Инициализация
БД</a></li>
        </ol>
      </td>
    </tr>
  </table>

```

```

  <p>Инициализация БД должна запускаться один раз перед началом
работы с системой. Этот сценарий создает в БД коллекции и файлы,
необходимые для начала работы.</p>

```

[Оглавление](#)

```

</body>
</html>
(: ++++++ : )

```

### 7.3.5.7.2 Сценарий initdb.xql

Сценарий предназначен для инициализации БД перед работой с приложением. Он создает необходимые коллекции и файлы в БД.

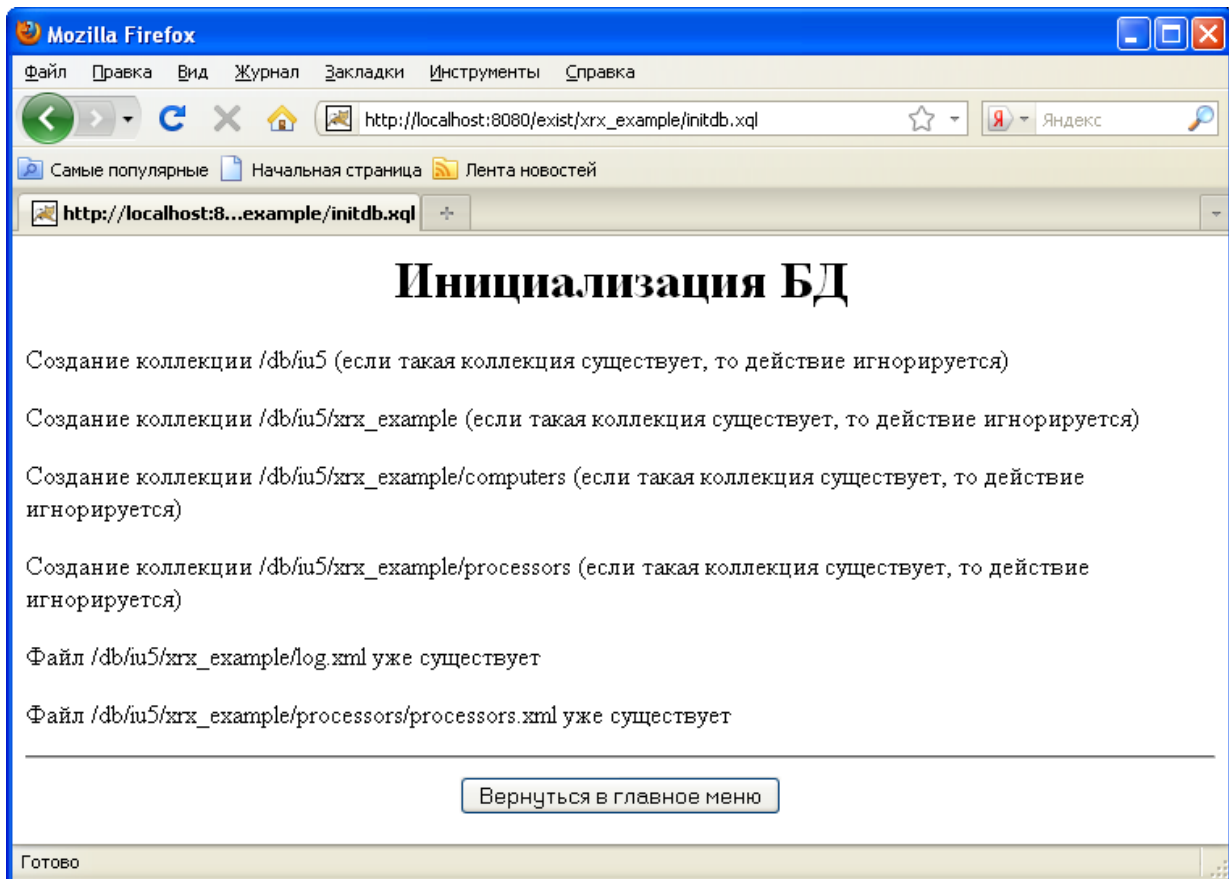


Рис. 7.16. Сценарий initdb.xql

Сценарий разработан таким образом, что его можно запускать многократно, все введенные данные при этом сохраняются.

Сценарий содержит пролог и основной запрос (локальных функций нет).

Пролог сценария:

```

xquery version "1.0";
(: Импорт стандартных модулей :)
declare namespace request="http://exist-db.org/xquery/request";
declare namespace response="http://exist-db.org/xquery/response";

```



```

declare namespace session="http://exist-db.org/xquery/session";
(: Импорт модулей приложения :)
import module namespace xrx_example =
"http://iu5.bmstu.ru/edu/xrx_example" at "module.xqm";
(: Формат в котором результат должен отображаться в браузере :)
declare option exist:serialize "method=xhtml media-type=text/html
indent=no";
(: Идентификаторы модулей для создания коллекций :)
declare variable $module_processor_id {"processor"};
declare variable $module_computer_id {"computer"};

```

### Основной запрос сценария:

```

(: ++++++ : )
(: Основной запрос :)
(: ++++++ : )
let

```

### В переменные запроса сохраняются следующие значения:

```

$Computer_CollectionName :=
xrx_example:PathURI_data_list_in_db_not_basepath($module_computer_id
),

```

### Название коллекции компьютеров.

```

$Processor_CollectionName :=
xrx_example:PathURI_data_list_in_db_not_basepath($module_processor_id
),

```

### Название коллекции процессоров.

```

$Processor_CollectionNameWithBasePath :=
xrx_example:PathURI_data_list_in_db($module_processor_id)

```

Полный путь к коллекции процессоров, используется для создания файла со списком процессоров.

```

return
(
  (<h1 align="center">Инициализация БД</h1>),
  (: Создание коллекции iu5 в коллекции /db :)

```

```

    xrx_example:InitDB_Create_Collection($xrx_example:BasePath_db,
    $xrx_example:BasePath_iu5),
    (: Создание коллекции xrx_example в коллекции /db/iu5/ :)
    xrx_example:InitDB_Create_Collection($xrx_example:BasePath_db_iu5,
    $xrx_example:BasePath_xrx_example),
    (: Создание коллекции computers в коллекции /db/iu5/xrx_example/
    :))
    xrx_example:InitDB_Create_Collection($xrx_example:BasePath,
    $Computer_CollectionName),
    (: Создание коллекции processors в коллекции /db/iu5/xrx_example/
    :))
    xrx_example:InitDB_Create_Collection($xrx_example:BasePath,
    $Processor_CollectionName),

```

С помощью функции `xrx_example:InitDB_Create_Collection` создается коллекция в БД. Первый параметр – название базовой коллекции, второй параметр – название новой коллекции, которая создается в базовой.

Если создаваемая коллекция уже существует, то она остается без изменений.

```

    (: Создание файла log.xml (для отладочных сообщений) в коллекции
    /db/iu5/xrx_example/ :)
    xrx_example:InitDB_Create_File($xrx_example:BasePath,
    $xrx_example:FileName_log, <root/>),
    (: Создание файла processors.xml (процессоры) в коллекции
    /db/iu5/xrx_example/processors/ :)

```

```

xrx_example:InitDB_Create_File($Processor_CollectionNameWithBasePath
, $xrx_example:FileName_processors, <processors/>),

```

С помощью функции `xrx_example:InitDB_Create_File` создается файл в заданной коллекции БД. Первый параметр – название коллекции, второй параметр – название создаваемого файла, третий параметр – XML-фрагмент, который сохраняется в файл.

Если файл уже существует, то он не создается заново.

```

(
<div>

```

```

<hr/>
<table align="center">
  <tr>
    {xrx_example:PathURI_menu()}
  </tr>
</table>
</div>
)
)

```

Формирование кнопки возврата в главное меню.

### 7.3.5.7.3 Таблица стилей style.css

Основная таблица стилей, которая применяется ко всем динамически генерируемым HTML-документам.

Таблица стилей составлена в соответствии со стандартом CSS [CSS, 2009].

Так как таблица стилей добавляется в XQuery-сценарий с помощью технологии XInclude, то она «обрамляется» корневым тэгом и оформляется в виде XML-документа.

Для того, чтобы избежать ошибок разбора таблицы стилей, в начало таблицы стилей помещается пустое правило empty {}.

```

<?xml version="1.0" encoding="UTF-8"?>
<css>
/* пустое правило, чтобы не было ошибок разбора таблицы стилей */
empty {}

h1 {
  text-align: center;
  font-size: 1.5em;
  font-weight: normal;
  font-family: Arial;
  color: white;
  background-color: #0990EE;
  margin-left: 0px;
  margin-bottom: 15px;

```

```

margin-top: 0px;
padding-left: 5px;
padding-right: 5px;
padding-top: 5px;
padding-bottom: 5px;
-moz-border-radius: 15px;
-webkit-border-radius: 15px;
}

```

```

h2 {
    text-align: center;
    font-size: 1em;
    font-weight: normal;
    font-family: Arial;
    color: white;
    background-color: #0990EE;
    margin-left: 0px;
    margin-bottom: 15px;
    margin-top: 0px;
    padding-left: 5px;
    padding-right: 5px;
    padding-top: 5px;
    padding-bottom: 5px;
    -moz-border-radius: 10px;
    -webkit-border-radius: 10px;
}

```

```

th, td
{
    padding: 5px;
}

```

```

.menu li
{
    margin-top: 15px;
}

```

```
}

a.menu, a.menu:visited {
    text-align: left;
    text-decoration: none;
    color: blue;
    font-size: 1em;
    font-weight: bold;
    font-family: Arial;
    margin-left: 5px;
    margin-right: 5px;
    margin-top: 5px;
    margin-bottom: 5px;
    padding-left: 10px;
    padding-right: 10px;
    padding-top: 10px;
    padding-bottom: 10px;
}

a.menu:hover {
    text-align: left;
    text-decoration: none;
    font-size: 1em;
    font-weight: bold;
    font-family: Arial;
    color: white;
    background-color: #0990EE;
    margin-left: 5px;
    margin-right: 5px;
    margin-top: 5px;
    margin-bottom: 5px;
    padding-left: 10px;
    padding-right: 10px;
    padding-top: 10px;
    padding-bottom: 10px;
}
```

```

-moz-border-radius: 15px;
-webkit-border-radius: 15px;
}
</css>

```

#### 7.3.5.7.4 Таблица стилей xforms.css

Таблица стилей, которая используется для форматирования XForms-форм.

```

<?xml version="1.0" encoding="UTF-8"?>
<css>
/* пустое правило, чтобы не было ошибок разбора таблицы стилей */
empty {}

/* Определение пространства имен для XForms */
@namespace xf url("http://www.w3.org/2002/xforms");

/* Задание параметров для элемента value, который соответствует
элементу textarea (value вложен в textarea) */
/* Связь с пространством имен через префикс xf */
xf|textarea xf|value
{
vertical-align: middle;
width : 300px;
height: 40px;
}
</css>

```

#### 7.3.5.7.5 Модуль module.xqm

XQuery-модуль содержит вспомогательные функции.

Модуль (в отличие от обычного сценария) не содержит основного запроса. Он содержит только пролог и библиотечные функции.

Далее приводится текст модуля с комментариями:

```
xquery version "1.0";
```

```
(: Наименование модуля :)
```

```
module namespace xrx_example =
"http://iu5.bmstu.ru/edu/xrx_example";
(: Пространство имен модуля - "http://iu5.bmstu.ru/edu/xrx_example"
В сценариях для подключения модуля должна использоваться команда
import module namespace префикс =
"http://iu5.bmstu.ru/edu/xrx_example" at "module.xqm";
Функции модуля вызываются с использованием префикса -
префикс:функция(параметры)
Префиксы могут быть произвольными (главное, чтобы совпадали
пространства имен).
В нашем примере в модуле и сценариях используется одинаковый
префикс - xrx_example. :)
```

```
(: Импорт стандартных модулей :)
declare namespace request="http://exist-db.org/xquery/request";
declare namespace session="http://exist-db.org/xquery/session";
declare namespace xdb="http://exist-db.org/xquery/xmldb";
```

```
(: Базовый путь к приложению :)
declare variable $xrx_example:BasePath {"/db/iu5/xrx/"};
(: Коллекции в базовом пути, используются при инициализации БД :)
declare variable $xrx_example:BasePath_db {"/db"};
declare variable $xrx_example:BasePath_iu5 {"iu5"};
declare variable $xrx_example:BasePath_db_iu5 {"/db/iu5/"};
declare variable $xrx_example:BasePath_xrx_example {"xrx"};
```

```
(: Наименование файла данных для процессоров :)
declare variable $xrx_example:FileName_processors
{"processors.xml"};
(: Наименование файла для отладочных сообщений :)
declare variable $xrx_example:FileName_log {"log.xml"};
```

```
(: Наименование основного сценария :)
declare variable $xrx_example:MainScript {"index.xql"};
```

```
(: Базовый путь к XSLTForms :)

declare variable $xrx_example:BaseXSLTFormsPath {"
http://localhost:8080/exist/rest/db/iu5/xrx/xsltforms/"};

(: Путь к XSLT-преобразованию в XSLTForms :)
declare variable $xrx_example:XSLTFormsFile {
fn:concat($xrx_example:BaseXSLTFormsPath, "xsltforms.xml") };

(: ++++++ : )
(: Функции для работы с XForms :)
(: ++++++ : )

(: ++++++ : )
(: Преобразование XForms-формы в HTML с помощью XSLT-преобразования
:)
(: Не требуется специальных настроек eXist, не требуется
преобразований в браузере :)
(: ++++++ : )
declare function xrx_example:transform_xsltforms($xhtml as
node()) as node()*
{
(: Параметры преобразования :)
let
    $XSLTFile := xs:anyURI($xrx_example:XSLTFormsFile),
    (: В качестве параметра преобразования необходимо указать базовый
путь к XSLTForms,
    иначе преобразование генерирует неправильные пути в HTML
(неправильно генерируются пути к файлам JavaScript и т.д.) :)
    $params :=
        <parameters>
            <param name="baseuri"
value="{ $xrx_example:BaseXSLTFormsPath }"/>
        </parameters>
    (: XSLT-преобразование, возвращается HTML-документ :)
}
```



```

let $TransformRes := transform:transform($xhtml doc, $XSLTFile,
$params)
return $TransformRes
};

```

```

(: ++++++ : )
(: Функции для работы с коллекциями и сценариями : )
(: ++++++ : )

```

```

(: ++++++ : )
(: Переход на заданный URI : )
(: Позволяет из одного XQuery-сценария переходить на другой XQuery-
сценарий : )
(: ++++++ : )
declare function xrx_example:redirect($uri_str as xs:string) as
empty()
{
let $uri := xs:anyURI($uri_str)
return response:redirect-to($uri)
};

```

```

(: ++++++ : )
(: Возвращает путь к коллекции, соответствующей модулю : )
(: К названию модуля добавляется буква "s" : )
(: ++++++ : )
declare function
xrx_example:PathURI_data_list_in_db_not_basepath($module_id as
xs:string) as xs:string
{
let $Result := concat($module_id, "s")
return $Result
};

```

```

(: ++++++ : )
(: Возвращает полный путь к коллекции, соответствующей модулю : )

```

(: Базовый путь к приложению объединяется с результатом предыдущей функции :)

(: ++++++ :)

```
declare function xrx_example:PathURI_data_list_in_db($module_id as
xs:string) as xs:string
```

```
{
```

```
let $Result := concat($xrx_example:BasePath,
```

```
xrx_example:PathURI_data_list_in_db_not_basepath($module_id), "/")
```

```
return $Result
```

```
};
```

(: ++++++ :)

(: Название файла, содержащего справочник процессоров :)

(: ++++++ :)

```
declare function xrx_example:ProcessorsFileName() as xs:string
```

```
{
```

```
let $Result :=
```

```
concat(xrx_example:PathURI_data_list_in_db("processor"),
```

```
$xrx_example:FileName_processors)
```

```
return $Result
```

```
};
```

(: ++++++ :)

(: Выделение из URL части, содержащей название сценария :)

(: ++++++ :)

```
declare function xrx_example:FileNameFromURL($param as xs:string) as
xs:string
```

```
{
```

```
let $delim := "/"
```

```
return
```

(: Если текущий адрес содержит "/", то функция вызывается рекурсивно,

параметром является то, что стоит справа от "/". То есть от URL постепенно отделяются названия каталогов, которые соединены между собой "/".

Рекурсивный вызов является заменой цикла while.

```

:)
if(fn:contains($param,$delim))
    then (xrx_example:FileNameFromURL(fn:substring-
after($param,$delim)))
        (: Возвращается название сценария, которое не содержит "/" :)
    else ($param)
};

(: ++++++ :)
(: Получение текущего URL :)
(: ++++++ :)
declare function xrx_example:CurrentURL() as xs:string
{
let $Result := xs:string(session:encode-url(request:get-uri()))
return $Result
};

(: ++++++ :)
(: Получение названия сценария из текущего URL :)
(: ++++++ :)
declare function xrx_example:FileNameFromCurrentURL() as xs:string
{
let $Result := xrx_example:FileNameFromURL(xrx_example:CurrentURL())
return $Result
};

(: ++++++ :)
(: Получение базового URL :)
(: ++++++ :)
declare function xrx_example:BaseURL($url as xs:string) as xs:string
{
let
    $FileName := xrx_example:FileNameFromURL($url),
    $FNLen := string-length($FileName),

```

```

$Result := substring($url, 1, string-length($url)-$FNLen)
return $Result
};

```

```

(: ++++++ : )
(: Получение базового URL из текущего URL
request:get-url() возвращает полный адрес, начиная с http :)
(: ++++++ : )
declare function xrx_example:BaseCurrentURL() as xs:string
{
let $Result := xrx_example:BaseURL(request:get-url())
return $Result
};

```

```

(: ++++++ : )
(: Возвращает строку, содержащую версию модуля :)
(: Функция используется для генерации путей к сценариям.
Если указана версия модуля, то функция возвращает подчеркивание и
версию модуля,
иначе пустую строку :)
(: ++++++ : )
declare function xrx_example:module_ver_str($module_ver as
xs:string) as xs:string
{
if($module_ver != "")
then (concat("_", $module_ver))
else ("")
};

```

```

(: ++++++ : )
(: Возвращает URI сценария, который выводит список :)
(: ++++++ : )
declare function xrx_example:PathURI_list($module_id as xs:string,
$module_ver as xs:string) as xs:string
{

```

```

let $Result := concat("list_", $module_id,
xrx_example:module_ver_str($module_ver), ".xql")
return $Result
};

(: ++++++ : )
(: Возвращает URI сценария редактирования :)
(: ++++++ : )
declare function xrx_example:PathURI_edit($module_id as xs:string,
$module_ver as xs:string) as xs:string
{
let $Result := concat("edit_", $module_id,
xrx_example:module_ver_str($module_ver), ".xql")
return $Result
};

(: ++++++ : )
(: Возвращает URI сценария, который работает с данными :)
(: ++++++ : )
declare function xrx_example:PathURI_data($module_id as xs:string,
$module_ver as xs:string) as xs:string
{
let $Result := concat("data_", $module_id,
xrx_example:module_ver_str($module_ver), ".xql")
return $Result
};

(: ++++++ : )
(: Возвращает URI сценария, который сохраняет данные :)
(: ++++++ : )
declare function xrx_example:PathURI_data_save($module_id as
xs:string, $module_ver as xs:string) as xs:string
{
let $Result := concat(xrx_example:PathURI_data($module_id,
$module_ver), "?save=1")

```

```
return $Result
};
```

```
(: ++++++ :)  
(: Возвращает URI сценария, который подготавливает данные для  
сохранения :)  
(: ++++++ :)  
declare function xrx_example:PathURI_data_presave($module_id as  
xs:string, $module_ver as xs:string) as xs:string  
{  
let $Result := concat(xrx_example:PathURI_data($module_id,  
$module_ver), "?presave=1")  
return $Result  
};
```

```
(: ++++++ :)  
(: Возвращает URI сценария, который возвращает список данных :)  
(: ++++++ :)  
declare function xrx_example:PathURI_data_list($module_id as  
xs:string, $module_ver as xs:string) as xs:string  
{  
let $Result := concat(xrx_example:PathURI_data($module_id,  
$module_ver), "?list=1")  
return $Result  
};
```

```
(: ++++++ :)  
(: Возвращает URI сценария, который возвращает набор данных по  
умолчанию  
(Используется только с XForms, так как набор данных нужно  
генерировать в отдельном сценарии и передавать в форму) :)  
(: ++++++ :)  
declare function xrx_example:PathURI_data_edit($module_id as  
xs:string, $module_ver as xs:string, $idParam as xs:string,  
$newParam as xs:string) as xs:string
```

```

{
let $MainPath := concat("data_", $module_id,
xrx_example:module_ver_str($module_ver), ".xql?")
return
  if($idParam != "0")
    then ( concat($MainPath, "id=", $idParam) )
    else ( concat($MainPath, "new=1") )
};

(: ++++++ : )
(: Возвращает id параметр для формы редактирования (используется без
XForms) :)
(: ++++++ : )
declare function xrx_example:PathURI_make_id_param($module_id as
xs:string, $idParam as xs:string, $newParam as xs:string) as
xs:string
{
if($idParam != "0")
  then ($idParam)
  else (xs:string("new"))
};

(: ++++++ : )
(: Генерация формы на переход к просмотру списка :)
(: ++++++ : )
declare function xrx_example:PathURI_list_form($module_id as
xs:string, $module_ver as xs:string) as node()
{
<form method="post" action="{xrx_example:PathURI_list($module_id,
$module_ver)}">
  <td>
    <button type="submit">Вернуться к списку</button>
  </td>
</form>
};

```

```

(: ++++++ : )
(: Генерация формы для возврата в главное меню : )
(: ++++++ : )
declare function xrx_example:PathURI_menu() as node()
{
<form method="post" action="{ $xrx_example:MainScript }">
  <td>
    <button type="submit">Вернуться в главное меню</button>
  </td>
</form>
};

(: ++++++ : )
(: Генерация формы на добавление элемента по названию модуля : )
(: ++++++ : )
declare function xrx_example:PathURI_edit_new_form($module_id as
xs:string, $module_ver as xs:string) as node()
{
<form method="post" action="{xrx_example:PathURI_edit($module_id,
$module_ver)}">
  <td>
    <input type="hidden" name="new" value="1" />
    <button type="submit">Добавить новый элемент</button>
  </td>
</form>
};

(: ++++++ : )
(: Генерация формы на редактирование элемента по названию модуля : )
(: ++++++ : )
declare function xrx_example:PathURI_edit_id_form($module_id as
xs:string, $module_ver as xs:string, $id as xs:string) as node()
{

```



```

<form method="post" action="{xrx_example:PathURI_edit($module_id,
$module_ver)}">
  <td>
    <input type="hidden" name="id" value="{ $id}" />
    <button type="submit"
title="Редактировать">Редактировать</button>
  </td>
</form>
};

```

```

(: ++++++ : )
(: Генерация формы на удаление элемента по названию модуля :)
(: ++++++ : )
declare function xrx_example:PathURI_delete_id_form($module_id as
xs:string, $module_ver as xs:string, $id as xs:string) as node()
{
<form method="post" action="{xrx_example:PathURI_data($module_id,
$module_ver)}">
  <td>
    <input type="hidden" name="deleteid" value="{ $id}" />
    <button type="submit" title="Удалить">Удалить</button>
  </td>
</form>
};

```

```

(: ++++++ : )
(: Функции для работы с файлами CSS :)
(: ++++++ : )

(: ++++++ : )
(: Возвращает путь к основному файлу CSS :)
(: ++++++ : )
declare function xrx_example:PathURI_current_css() as xs:string
{

```

```

let $Result := concat(xrx_example:BaseCurrentURL(),
"styles/style.css")
return $Result
};

```

```

(: ++++++ : )
(: Возвращает путь к файлу CSS для XForms :)
(: ++++++ : )
declare function xrx_example:PathURI_xforms_css() as xs:string
{
let $Result := concat(xrx_example:BaseCurrentURL(),
"styles/xforms.css")
return $Result
};

```

```

(: ++++++ : )
(: Выдача короткого сообщения об ошибке :)
(: ++++++ : )
declare function xrx_example:ShowErrorButton($Error_message as
xs:string, $Button_text as xs:string, $Button_uri as xs:string) as
node()
{
let $CurrentCssURI := xrx_example:PathURI_current_css()
return
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8"/>
    <style type="text/css">
      <!-- Добавление стилей CSS с помощью XInclude -->
      <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
href="{ $CurrentCssURI }"/>
    </style>
  </head>
  <body>

```

```

<table align="center" border="1">
<tr>
  <td>
    <h2>Ошибка</h2>
  </td>
</tr>
<tr>
  <td>
    <p>{$Error_message}</p>
  </td>
</tr>
{
if($Button_text != "" and $Button_uri != "")
then
(
<tr>
  <td align="center">
    <form method="post" action="{ $Button_uri }">
      <button type="submit">{$Button_text}</button>
    </form>
  </td>
</tr>
)
else ( )
  }
</table>
</body>
</html>
};

```

```
(: ++++++ :)
```

```
(: Выдача сообщения об ошибке при работе с формой :)
```

```
(: ++++++ :)
```

```

declare function xrx_example:ShowError($module_id as xs:string,
$module_ver as xs:string, $message as xs:string, $return_form as
xs:boolean, $Errors as node()*) as node()
{
let $CurrentCssURI := xrx_example:PathURI_current_css()
return
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8"/>
    <style type="text/css">
      <!-- Добавление стилей CSS с помощью XInclude -->
      <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
href="{ $CurrentCssURI }"/>
    </style>
  </head>
  <body>
    <table align="center" border="1">
      <tr>
        <td>
          <h2>{$message}</h2>
        </td>
      </tr>
      <tr>
        <td>
          <p><b>Список ошибок:</b></p>
          <div>{$Errors}</div>
        </td>
      </tr>
      <tr>
        <td align="center">
          <form method="post"
action="{xrx_example:PathURI_list($module_id, $module_ver)}">
            {
              if($return_form)

```

```

        then
        (
            <button type="button" onclick="history.back()">Возврат к
редактированию</button>
        )
        else()
        }
        {
        if($module_id != "")
        then
        (
            <button type="submit">Отменить</button>
        )
        else ()
        }
        </form>
    </td>
</tr>
</table>
</body>
</html>
};

```

```
(: ++++++ :)
```

```
(: Выдача сообщения об ошибке при работе с формой :)
```

```
(: ++++++ :)
```

```

declare function xrx_example:ShowErrorReturnList($module_id as
xs:string, $module_ver as xs:string, $message as xs:string, $Errors
as node()*) as node()
{
let $Result := xrx_example:ShowError($module_id, $module_ver,
$message, fn:false(), $Errors)
return $Result
};

```

```
(: ++++++ :)
```

```
(: Выдача сообщения об ошибке и возврат в список :)
```

```
(: ++++++ :)
```

```
declare function xrx_example:ShowErrorReturnForm($module_id as
xs:string, $module_ver as xs:string, $message as xs:string, $Errors
as node()*) as node()
{
let $Result := xrx_example:ShowError($module_id, $module_ver,
$message, fn:true(), $Errors)
return $Result
};
```

```
(: ++++++ :)
```

```
(: Функции для работы с идентификаторами :)
```

```
(: ++++++ :)
```

```
(: ++++++ :)
```

```
(: Создание уникального идентификатора на основе текущей даты :)
```

```
(: ++++++ :)
```

```
declare function xrx_example:MakeUniqueNow() as xs:string
{
let
    $now := fn:current-dateTime(),
    $year := xs:string(fn:year-from-dateTime($now)),
    $month := xs:string(fn:month-from-dateTime($now)),
    $day := xs:string(fn:day-from-dateTime($now)),
    $h := xs:string(fn:hours-from-dateTime($now)),
    $m := xs:string(fn:minutes-from-dateTime($now)),
    $s := xs:string(fn:seconds-from-dateTime($now)),
    $sr := replace($s, "\.", "F"),
    $Result := concat($year, "_", $month, "_", $day, "_", $h, "_", $m,
"_", $sr)
return $Result
};
```

```
(: ++++++ :)
```

(: Создание уникального идентификатора :)

(: Идентификатор создается как объединение стандартного уникального идентификатора (функция util:uuid()) и уникального идентификатора на основе текущей даты (функция xrx\_example:MakeUniqueNow()) :)

```
(: ++++++ :)
```

```
declare function xrx_example:MakeUniqueName() as xs:string
{
let
    $id := util:uuid(),
    $nowstr := xrx_example:MakeUniqueNow(),
    $Result := concat($id, "_" , $nowstr)
return $Result
};
```

```
(: ++++++ :)
```

(: Создание названия файла на основе уникального идентификатора :)

```
(: ++++++ :)
```

```
declare function xrx_example:MakeUniqueFileNameExtXml($id as
xs:string) as xs:string
{
    let $Result := concat($id, ".xml")
    return $Result
};
```

```
(: ++++++ :)
```

(: Функции для работы с данными и коллекциями :)

```
(: ++++++ :)
```

```
(: ++++++ :)
```

(: Количество элементов в модуле :)

(: Функция работает и в том случае, когда элементы хранятся в отдельных файлах,

и в том случае, когда элементы хранятся в одном файле.

Это достигается за счет использования запроса вида

"коллекция//название элемента".

Выражение "//" позволяет "пропустить" название файла, корневой элемент файла и т.д. :)

```
(: ++++++ :)
```

```
declare function xrx_example:ModuleDataCount($module_id as
xs:string, $module_ver as xs:string) as xs:integer
{
  (: Формирование запроса, который возвращает количество элементов,
  соответствующих модулю, в коллекции модуля :)
  let $CountText := concat(
    " let $cnt := ",
    "   collection('", xrx_example:PathURI_data_list_in_db($module_id),
    "')//", $module_id, " ",
    " return count($cnt)")
  (: Выполнение запроса :)
  return util:eval($CountText)
};
```

```
(: ++++++ :)
```

```
(: Сохранение данных в коллекции, если файл существует, то он
предварительно удаляется, это делает функция xdb:store :)
(: Для того, чтобы выполнять действия с коллекциями и файлами,
необходимо предварительно подключиться к БД с правами
администратора, это делает функция xrx_example:login() :)
(: ++++++ :)
```

```
declare function xrx_example:SaveFileInCollection($CollectionName as
xs:string, $id as xs:string, $Data as node())
{
  let
    $isLoggedIn := xrx_example:login(),
    $FileName := xrx_example:MakeUniqueFileNameExtXml($id)
  return xdb:store($CollectionName, $FileName, $Data)
};
```



```
(: ++++++ :)
```

(: Создание файла с данными в указанной коллекции, выдача сообщения :)

```
(: Если файл существует, то он не создается заново :)
```

(: Функция используется при инициализации БД :)

```
(: ++++++ :)
```

```
declare function xrx_example:InitDB_Create_File($CollectionName as
xs:string, $FileName as xs:string, $Data as node()) as node()*
{
let
    $isLoggedIn := xrx_example:login(),
    $CollectionFileName := concat($CollectionName, $FileName)
return
    if (doc-available($CollectionFileName))
    then
        (: Файл уже существует, выдача сообщения :)
        (
            <p>Файл {$CollectionFileName} уже существует</p>
        )
    else
        (: Файл не существует, создание файла :)
        (
            let $NewFileName := xdb:store($CollectionName, $FileName, $Data)
            return <p>Создание файла {$CollectionFileName}</p>
        )
};
```

```
(: ++++++ :)
```

(: Создание коллекции и выдача сообщения :)

(: Функция используется при инициализации БД.

Если коллекция уже существует, то коллекция не создается,  
ошибка не возникает :)

```
(: ++++++ :)
```

```

declare function
xrx_example:InitDB_Create_Collection($ParentCollectionName as
xs:string, $CollectionName as xs:string) as node()*
{
let
    $isLoggedIn := xrx_example:login(),
    $NewCol := xdb:create-collection($ParentCollectionName,
$CollectionName)
return <p>Создание коллекции {$NewCol} (если такая коллекция
существует, то действие игнорируется)</p>
};

(: ++++++ : )
(: Удаление файла из коллекции :)
(: ++++++ : )
declare function xrx_example>DeleteFileInCollection($CollectionName
as xs:string, $id as xs:string) as node()*
{
let
    $isLoggedIn := xrx_example:login(),
    $FileName := xrx_example:MakeUniqueFileNameExtXml($id)
return xdb:remove($CollectionName, $FileName)
};

(: ++++++ : )
(: Вход в БД пользователем, предназначенным для работы с БД :)
(: ++++++ : )
declare function xrx_example:login()
{
    xdb:register-database("org.exist.xmldb.DatabaseImpl", true()),
let
    $User := "admin",
    $Password := ""
return xdb:login("xmldb:exist:///db", $User, $Password)
};

```

```
(: ++++++ :)
```

```
(: Логирование :)
```

```
(: ++++++ :)
```

```
declare function xrx_example:log($Data as node(*)
{
let
    $LogFileName := concat($xrx_example:BasePath, "log.xml"),
    $LogPoint := doc($LogFileName)//root
    return update insert <data dt="{xs:string(fn:current-
dateTime())}">{$Data}</data> into $LogPoint
};
```

## 7.4 Дальнейшее улучшение примера

В приведенном примере XRX-приложения рассмотрены лишь основы разработки XRX-приложений.

В частности, не рассмотрены следующие вопросы:

1. Определение фильтра, чтобы данные в списке выводились в соответствии с заданными критериями.
2. Определение сортировки для списка данных, чтобы пользователь мог сортировать данные по заданному столбцу.
3. Постраничная выдача данных в списке при задании количества записей на странице, формирование кнопок перехода к предыдущей (следующей) странице.
4. Индексирование XML-данных в eXist.

При создании больших XRX-приложений вопросам индексирования необходимо уделить внимание, так как правильное или неправильное индексирование влияет на производительность приложения.

В документации eXist есть отдельное руководство по созданию индексов.

## **7.5 Материалы для дальнейшего изучения**

Рекомендуется ознакомиться с руководством в стандартной документации eXist по разработке XRX-приложений – «A Beginners Guide to XRX with eXist».

## **7.6 Контрольные вопросы**

1. В чем разница между архитектурой «классического» веб-приложения и архитектурой XRX?
2. Каким образом может применяться технология XForms в архитектуре XRX?
3. Каковы основные ограничения архитектуры XRX?
4. Каким образом можно отобразить инфологическую модель данных в модель данных СУБД «eXist»?
5. В чем особенность использования HTML-форм и XForms-форм в XRX-приложениях? Какой способ Вам кажется более удобным?

## Источники

1. [CSS, 2009] Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. W3C Candidate Recommendation 08 September 2009. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/CSS2/> – Загл. с экрана.
2. [Fielding, 2000] Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures, 2000. [электронный ресурс] – Режим доступа: <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm> – Загл. с экрана.
3. [MVC, 2007] Рогачев С. Обобщенный Model-View-Controller. [электронный ресурс] – Режим доступа: <http://www.rsdn.ru/article/patterns/generic-mvc.xml> – Загл. с экрана.
4. [XDM, 2007] XQuery 1.0 and XPath 2.0 Data Model (XDM), 2007. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xpath-datamodel/> – Загл. с экрана.
5. [XForms, 2009] XForms 1.1. W3C Recommendation 20 October 2009. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xforms/> – Загл. с экрана.
6. [XInclude, 2006] XML Inclusions (XInclude) Version 1.0 (Second Edition). W3C Recommendation 15 November 2006. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xinclude/> – Загл. с экрана.
7. [XLink, 2010] XML Linking Language (XLink) Version 1.1. W3C Recommendation 06 May 2010. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xlink11/> – Загл. с экрана.
8. [XML, 2000] 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.

9. [XML Schema Part 0 Primer, 2004] XML Schema Part 0: Primer Second Edition, 2004. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xmlschema-0/> – Загл. с экрана.
10. [XML Schema Part 1 Structures, 2004] XML Schema Part 1: Structures Second Edition, 2004. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xmlschema-1/> – Загл. с экрана.
11. [XML Schema Part 2 Datatypes, 2004] XML Schema Part 2: Datatypes Second Edition, 2004. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xmlschema-2/> – Загл. с экрана.
12. [XPath, 1999] Язык XML Path (XPath) версия 1.0, 1999. [электронный ресурс] – Режим доступа: <http://www.rol.ru/news/it/helpdesk/xpath01.htm> – Загл. с экрана.
13. [XPath, 2007] XML Path Language (XPath) 2.0, 2007. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xpath20/> – Загл. с экрана.
14. [XPath, 2014] XML Path Language (XPath) 3.0, 2014. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xpath-30/> – Загл. с экрана.
15. [XPointer Framework, 2003] XPointer Framework. W3C Recommendation 25 March 2003. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xptr-framework/> – Загл. с экрана.
16. [XPointer element() Scheme, 2003] XPointer element() Scheme. W3C Recommendation 25 March 2003. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xptr-element/> – Загл. с экрана.
17. [XPointer xmlns() Scheme, 2003] XPointer xmlns() Scheme. W3C Recommendation 25 March 2003. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xptr-xmlns/> – Загл. с экрана.
18. [XPointer xpointer() Scheme, 2002] XPointer xpointer() Scheme. W3C Working Draft 19 December 2002. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xptr-xpointer/> – Загл. с экрана.

- 19.[XQuery, 2007] XQuery 1.0: An XML Query Language, 2007.  
[электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xquery/> – Загл. с экрана.
- 20.[XQuery, 2014] XQuery 3.0: An XML Query Language, 2014.  
[электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xquery-30/> – Загл. с экрана.
- 21.[XQueryX, 2014] XQueryX 3.0: W3C Recommendation 08 April 2014.  
[электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xqueryx-30/> – Загл. с экрана.
- 22.[XQuery Functions, 2007] XQuery 1.0 and XPath 2.0 Functions and Operators, 2007. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xpath-functions/> – Загл. с экрана.
- 23.[XQuery Functions, 2014] XPath and XQuery Functions and Operators 3.0, 2014. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xpath-functions-30/> – Загл. с экрана.
- 24.[XQuery Update, 2011] XQuery Update Facility 1.0, 2011. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xquery-update-10/> – Загл. с экрана.
- 25.[XQuery руководство, 2005] XML: XQuery от экспертов. Руководство по языку запросов. Под ред. Г. Каца, КУДИЦ-Образ, 2005. – 480 с.
26. [XSLT, 1999] Язык преобразований XSL (XSLT) версия 1.0, 1999. [электронный ресурс] – Режим доступа: <http://www.rol.ru/news/it/helpdesk/xslt01.htm> – Загл. с экрана.
- 27.[XSLT, 2007] XSL Transformations (XSLT) Version 2.0, 2007. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xslt20/> – Загл. с экрана.
- 28.[XSLT, 2013] XSL Transformations (XSLT) Version 3.0, 2013. [электронный ресурс] – Режим доступа: <http://www.w3.org/TR/xslt-30/> – Загл. с экрана.

- 29.[Бурэ, 2002] Р. Бурэ. XML и базы данных, 2002. [электронный ресурс] – Режим доступа: [http://xml.nsu.ru/extra/database\\_0.xml](http://xml.nsu.ru/extra/database_0.xml) – Загл. с экрана.
- 30.[Валиков, 2002] Валиков А.Н. Технология XSLT. – СПб.: БХВ-Петербург, 2002. – 544 с.
- 31.[Мангано, 2008] Мангано С. XSLT. Сборник рецептов. – М.: ДМК Пресс, СПб.: БХВ-Петербург, 2008. – 864 с.
- 32.[Ширшов, 2003] Ширшов А. Использование XML совместно с SQL. [электронный ресурс] – Режим доступа: <http://www.rsdn.ru/article/db/xmlsql.xml> – Загл. с экрана.