Лабораторные работы 1-6
XML
XML-DOM

Оглавление

Введение

Настоящие лабораторные работы выполняются в рамках дисциплины “Корпоративные информационные системы” для подготовки дипломированных специалистов по специальности “Управление в технических системах” и бакалавровпо направлению “Информационные системы и технологии”.

Целью лабораторных работ является ознакомление со стандартами языка разметки XML, создание и просмотр простых XML-документов, а также ознакомление со смежными XML технологиями: языком таблиц стилей XSL, преобразованием документов в различные форматы путём XSL-трансформации, языком схем DTD и объектной моделью документа XML –DOM.

В ходе выполнения каждой лабораторной работы студентом (бригадой студентов) выполняются следующие действия: изучение содержания работы, ознакомление с заданием, работа по заданию и проверка результатов при помощи браузера или программы”XSL-процессор версии 2.0”, получение и сохранение результатов на диске, самостоятельная распечатка результатов и их анализ, подготовка отчета.

Лабораторная работа №1
«Способы создания XML-документов. Проверка документа на правильность построения. Поиск ошибок в неправильно построенном документе»

1 Цели лабораторной работы

Целью работы является получение сведений о способах создания XML-документов, проверки на правильность их построения, типичных ошибках и их поиску.

2 Содержание работы

- Способы создания и проверки на правильность построения XML-документов
- Поиск и устранение типичных ошибок
- Самостоятельная работа по созданию документа (устранению ошибок)

3 Порядок выполнения работы

Способы создания и проверки на правильность построения XML-документов

Процесс создания XML документа можно разбить на 2 этапа:

- Написание содержимого документа
- Проверка структуры документа

В простейшем случае для написания содержимого документа подойдёт обычный текстовый редактор. Например, текстовый редактор «Блокнот» (Notepad), который встроен в операционную систему Windows.

В окне редактора нужно набрать содержимое документа. Пример такого документа показан на рисунке 1.1

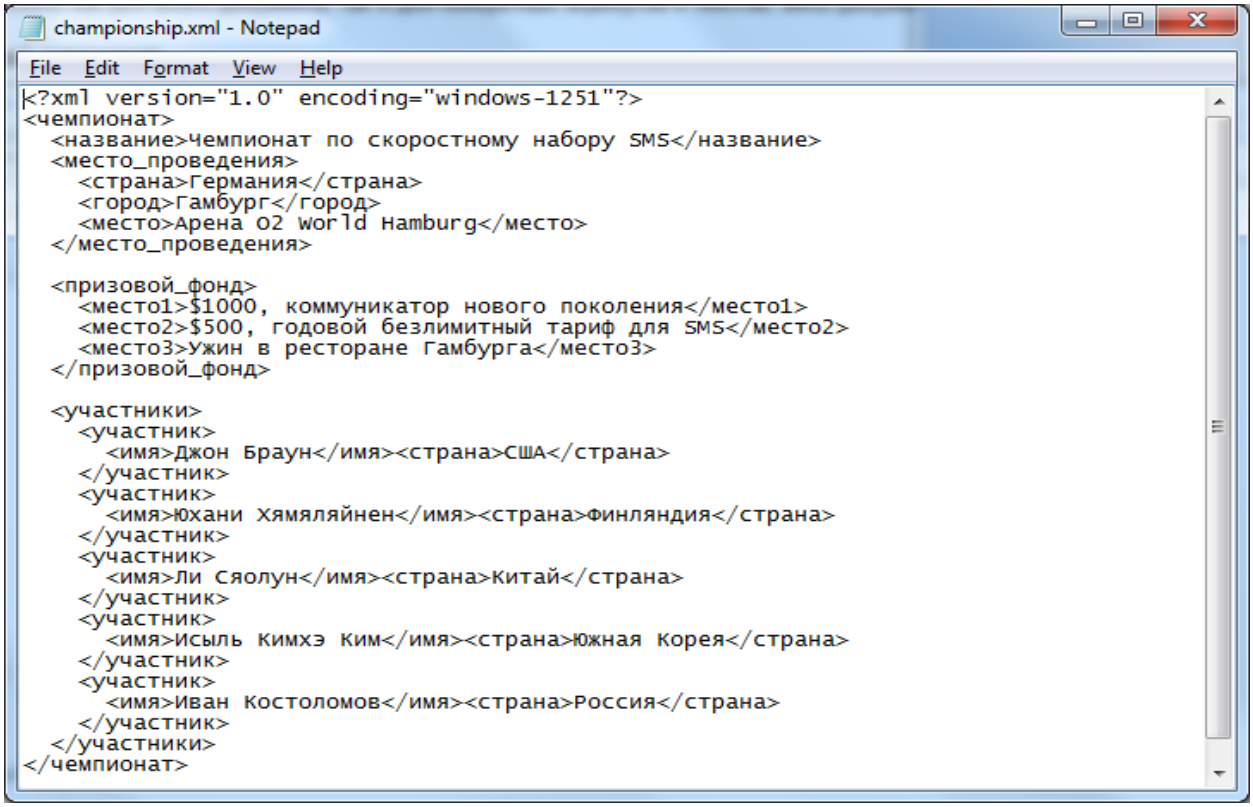


Рисунок 1.1. Документ XML в редакторе «Блокнот».

При сохранении документа часто используют расширение **xml.** Однако при создании документа Вы, по сути, создаёте новый формат. Необходимо учитывать тот факт, что документов, "написанных на XML", не может быть в принципе - в любом случае авторы документа для его разметки используют основанный на стандарте XML (т.н. XML-производный) язык, но не сам XML. Поэтому при сохранении созданного файла можно выбрать для него какое-то подходящее названию расширение.

Полученный файл необходимо проверить на правильность. Стандартом определены два уровня правильности документа XML:

- *Правильно построенный* (Well-formed). Правильно построенный документ соответствует всем общим правилам синтаксиса XML, применимым к любому XML-документу. И если, например, начальный тег не имеет соответствующего ему конечного тега, то это *неправильно построенный* документ XML. Документ, который неправильно построен, не может считаться документом XML; XML-процессор (парсер) не должен обрабатывать его обычным образом и обязан классифицировать ситуацию как *фатальная ошибка*.

• *Действительный* (Valid). Действительный документ дополнительно соответствует некоторым семантическим правилам. Это более строгая дополнительная проверка корректности документа на соответствие заранее определённым, но уже внешним правилам, в целях минимизации количества

ошибок, например, структуры и состава данного, конкретного документа или семейства документов. Эти правила могут быть разработаны как самим пользователем, так и сторонними разработчиками, например, разработчиками словарей или стандартов обмена данными. Обычно такие правила хранятся в специальных файлах — схемах, где самым подробным образом описана структура документа, все допустимые названия элементов, атрибутов и многое другое. И если документ, например, содержит не определённое заранее в схемах название элемента, то XML-документ считается *недействительным*; проверяющий XML-процессор (валидатор) при проверке на соответствие правилам и схемам обязан (по выбору пользователя) сообщить об ошибке.

• Данные два понятия не имеют достаточно устоявшегося стандартизированного перевода на русский язык, особенно понятие *valid*, которое можно также перевести, как *имеющий силу*, *правомерный*, *надёжный*, *годный*, или даже *проверенный на соответствие правилам, стандартам, законам*. Некоторые программисты применяют в обиходе устоявшуюся кальку «*Валидный*».

• В нашем примере мы не разрабатывали никаких семантических правил. Следовательно, мы имеем возможность проверить только структуру построения документа. Для этого можно воспользоваться встроенным в браузер MozillaFirefoxпарсером. Откроем созданный документ при помощи браузера, как показано на рисунке 1.2.

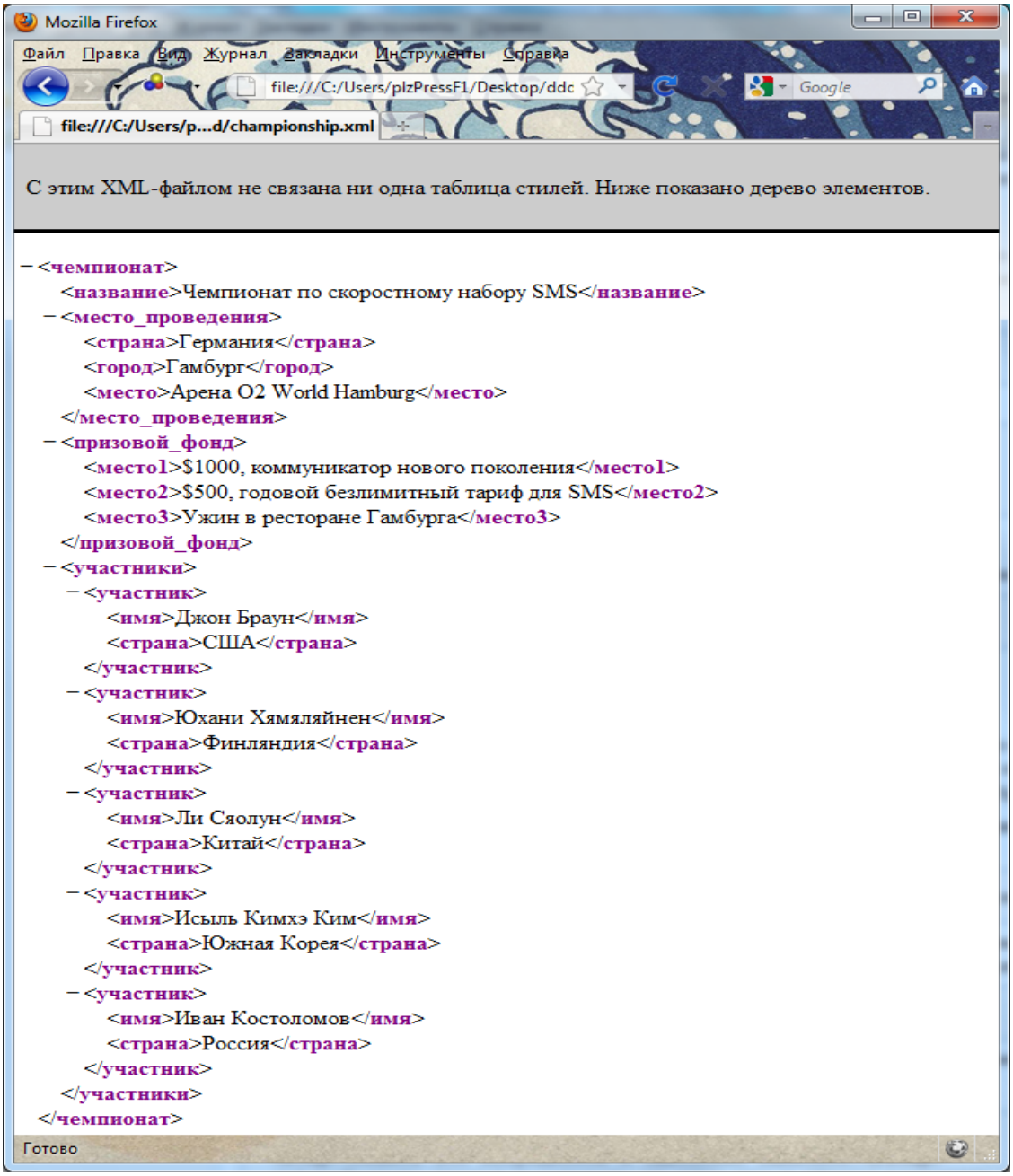


Рисунок 1.2. Отображение документа XML в браузере.

Программа сообщает, что с нашим XML-файлом не связана ни одна таблица стилей, и выводит дерево элементов на экран. Это означает, что документ имеет правильную структуру. В случае же ошибок при создании документа браузер выдаст следующую информацию, показанную на рисунке 1.3.

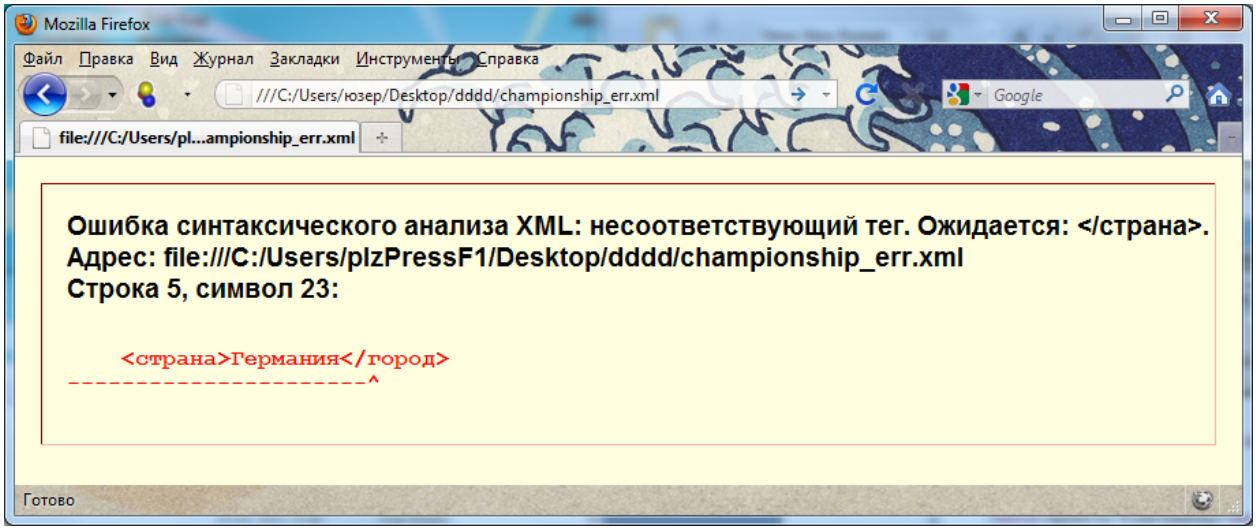


Рисунок 1.3. Сообщение об ошибке в браузере.

При этом проверка документа прекращается. Таким образом мы можем обнаружить лишь первую из возможных ошибок.

Кроме указанного способа создания XML документов, существует множество программ, при помощи которых можно создать документ и проверить его на правильность без использования программы-браузера. Примером такой программы является программа XMLPad. Программа распространяется бесплатно, получить информацию о программе можно по адресу: http://www.wmhelp.com/xmlpad3.htm.

При открытии программы и создании нового файла (пункт меню **File**, пункт **New**) появляется окно для выбора типа файла. Для создания файлов на этом уроке требуется выбрать **XML Document**. Окно с типами файлов показано на рисунке 1.4.

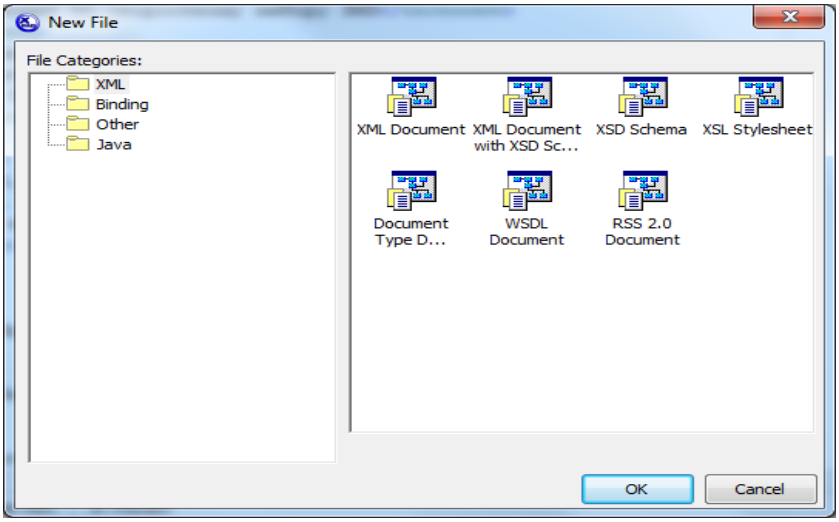


Рисунок 1.4. Выбор типа документа.

После выбора типа файла появится окно с выбором схемы структуры файла. Для создания простого файла требуется выбрать первый пункт **Notassigned**. Также в этом окне можно выбрать корневой тег (**Roottag**). Пример окна показан на рисунке 1.5

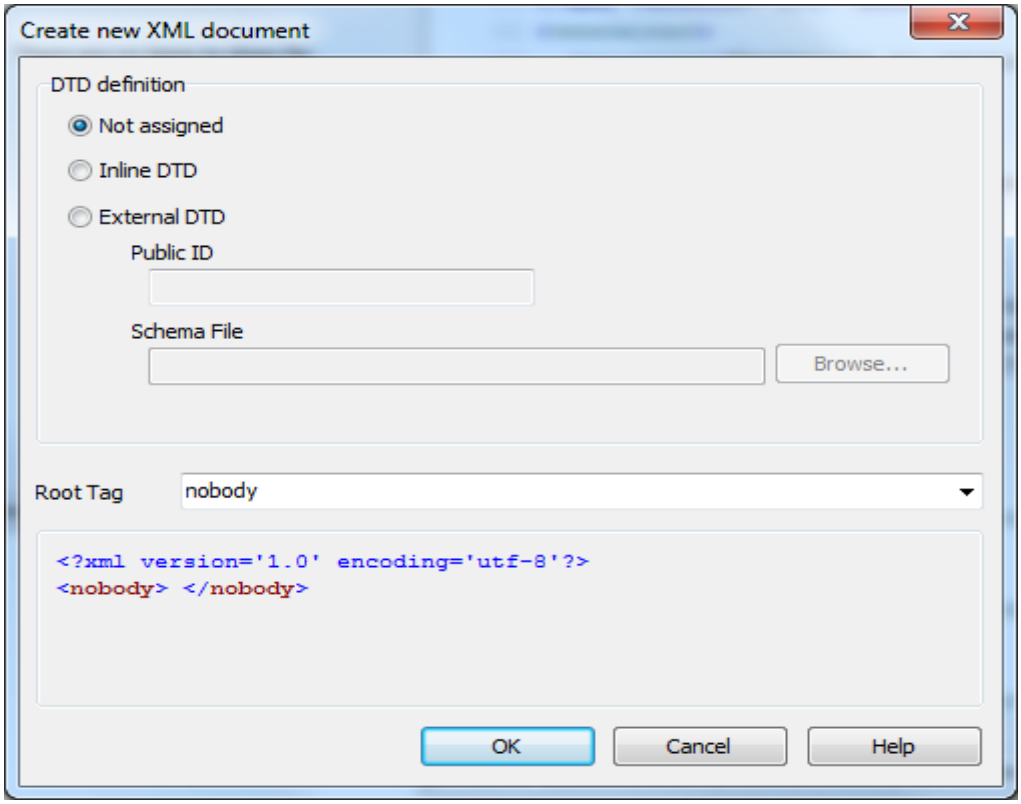


Рисунок 1.5. Окно создания нового документа.

По умолчанию документы создаются с кодировкой UTF-8. Для изменения кодировки открытого документа требуется изменить кодировку в окне **AdvancedSaveOptions**, которое можно вызвать из пункта меню **File** и пункта **AdvancedSaveOptions**. Пример окна показан на рисунке1.6.

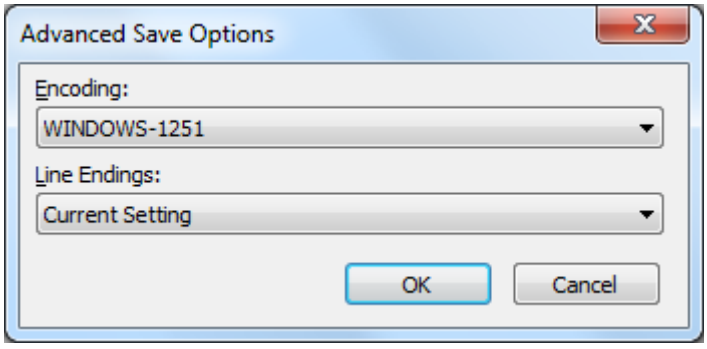


Рисунок 1.6. Окно выбора кодировки.

Важно: Установленная кодировка действует только на активный документ XML, открытый в программе.

Для проверки структуры документа требуется выбрать пункт меню **XML** и подпункт **Validate**. В случае корректности структуры программа не выдаст сообщений об ошибке. В противном случае поступит соответствующее сообщение, показанное на рисунке 1.7.

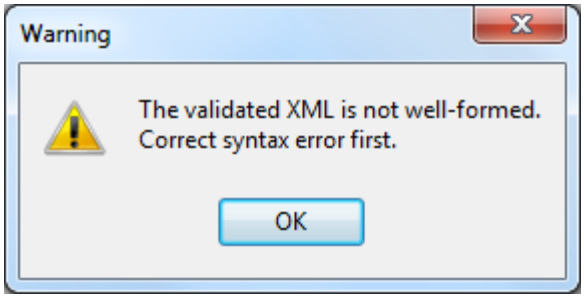


Рисунок 1.7. Сообщение об ошибке.

Также можно попытаться сохранить документ. При сохранении документа с некорректной структурой на экране появится предупреждение, показанное на рисунке 1.8.

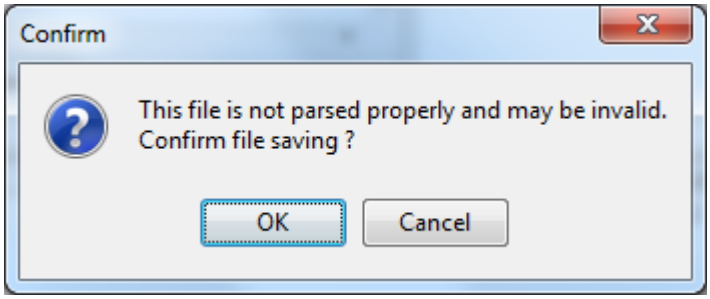


Рисунок 1.8. Предупреждение о некорректной структуре при сохранении.

Поиск и устранение типичных ошибок

В предыдущей части лабораторной работы мы рассмотрели некоторые способы проверки структуры XML-документа. В случае использования браузера MozillaFirefox мы могли получить сведения о месте ошибки (строка, символ) и её типе (например, *несоответствующий тег*). Программа XMLPad также обладает такой возможностью. Следует также отметить, что место ошибки называется весьма условно. Рассмотрим пример, показанный на рисунке 1.9.

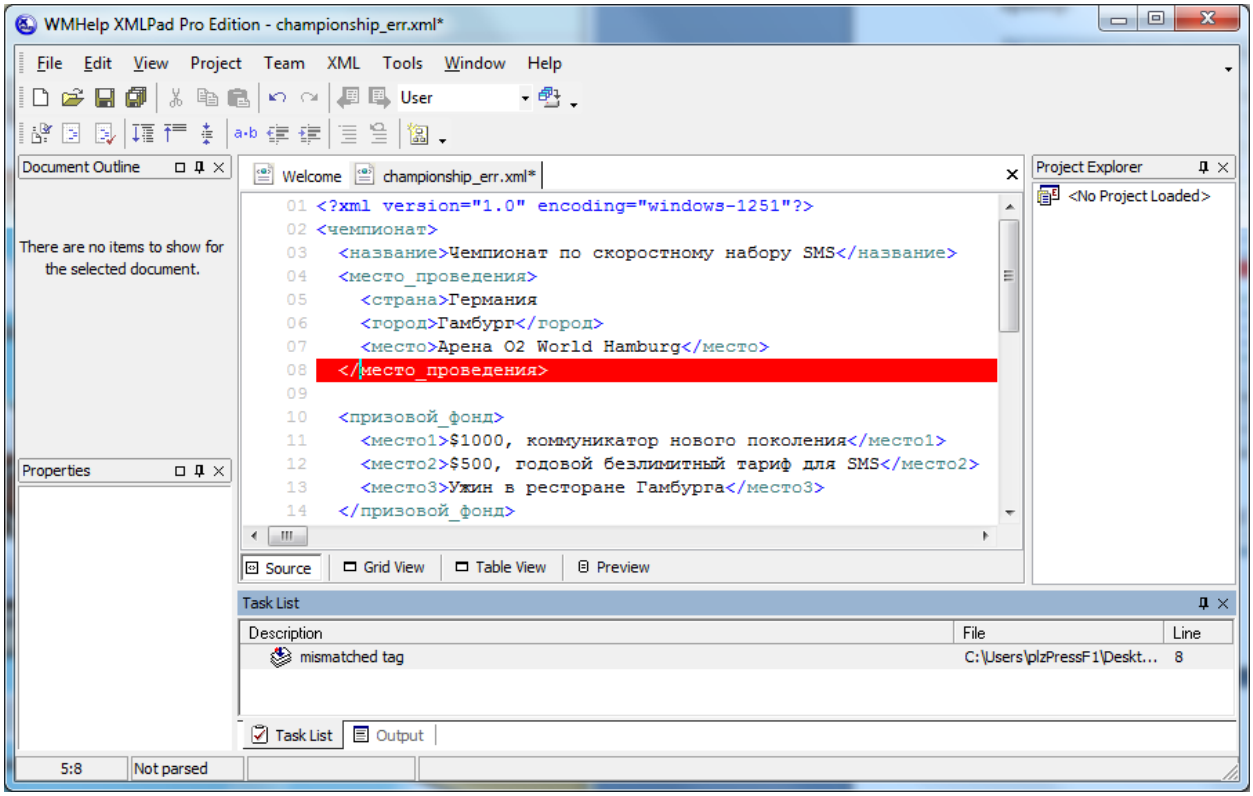


Рисунок 1.9. Главное окно программы, показывающее место ошибки.

В данном примере программа сообщила нам о пропущенном теге. Проанализировав строки выше, мы обнаруживаем отсутствие закрывающего тега для тега <страна>. Однако для составления синтаксически верного документа мы можем поставить недостающий тег в разные строки: в конец пятой, шестой, седьмой строки, а также в начало восьмой строки. Во всех случаях мы получим *правильно построенный* документ. Но логически верным вариантом будет размещение внутри тега <страна> содержимого, касающегося только названия страны без других вложенных тегов. Таким образом, мы разместим закрывающий тег в конце пятой строки.

Рассмотрим ещё некоторые примеры типичных ошибок, показанных на рисунках 1.10,1.11, 1.12.

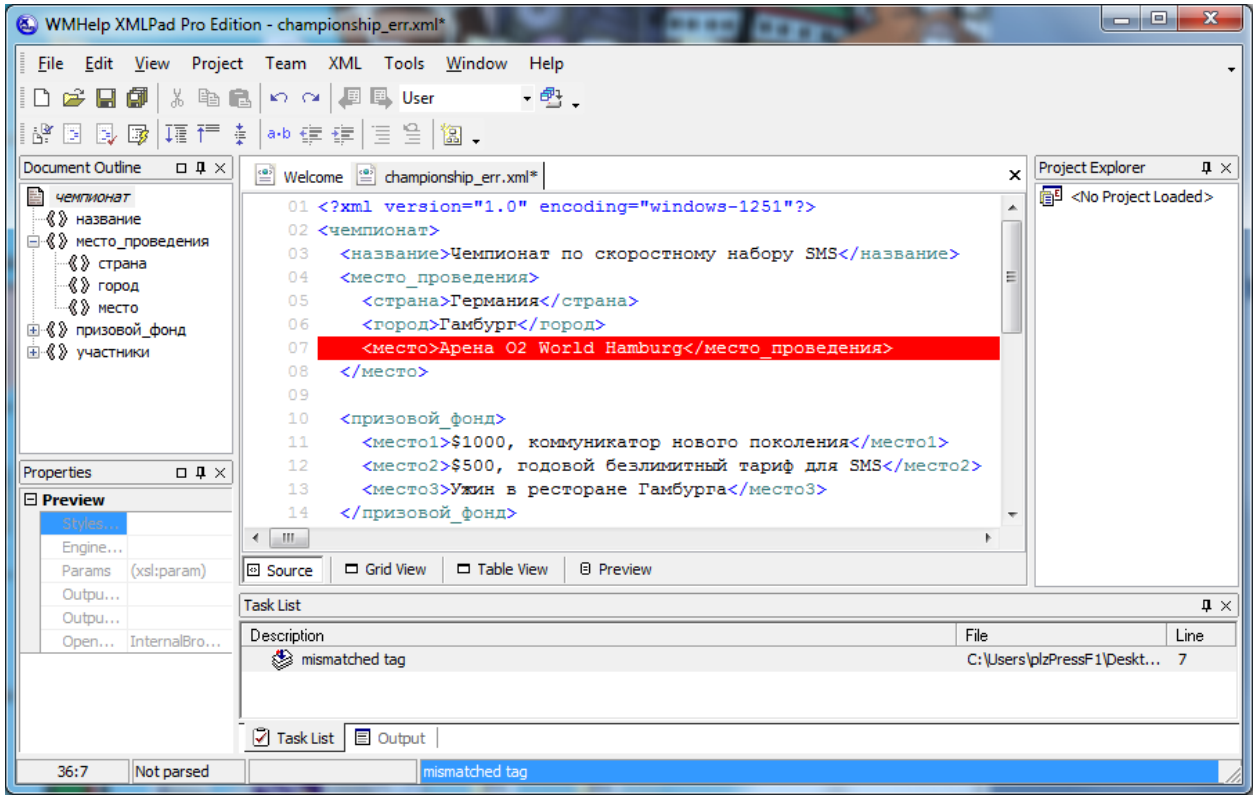


Рисунок 1.10. Главное окно программы, показывающее место ошибки.

В этом случае мы можем просто поставить отсутствующий закрывающий тег </место> прямо там, где нам это рекомендует программа. Но при следующей проверке мы обнаружим закрывающий тег </место> в восьмой строке, не имеющий парного открывающего тега. Простое удаление исправит ошибку. Но мы могли заметить это ещё на предыдущем шаге – закрывающие теги </место> и </место_проведения> были «перепутаны местами».

Порой случайные опечатки могут стать причиной ошибок, не заметных сразу.

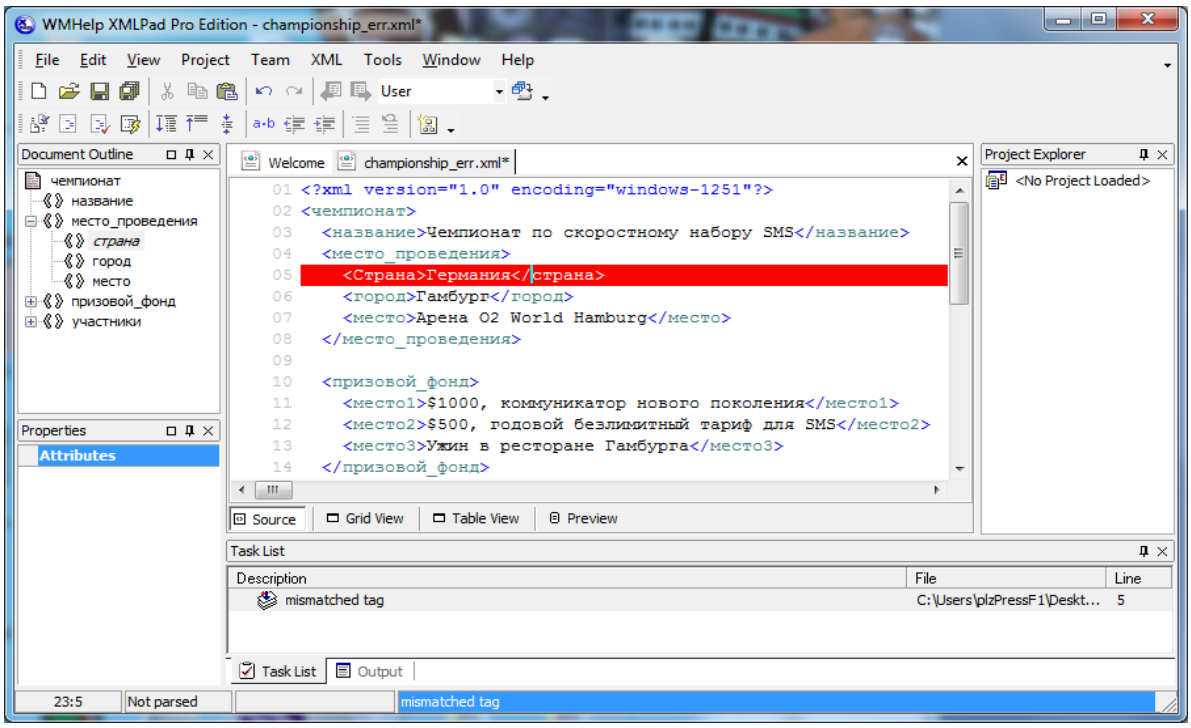


Рисунок 1.11. Главное окно программы, показывающее место ошибки.

Открывающий и закрывающий теги не идентичны с точки зрения регистра.

При копировании текста документа из программы MicrosoftWord может возникнуть следующая труднообнаружимая ошибка.

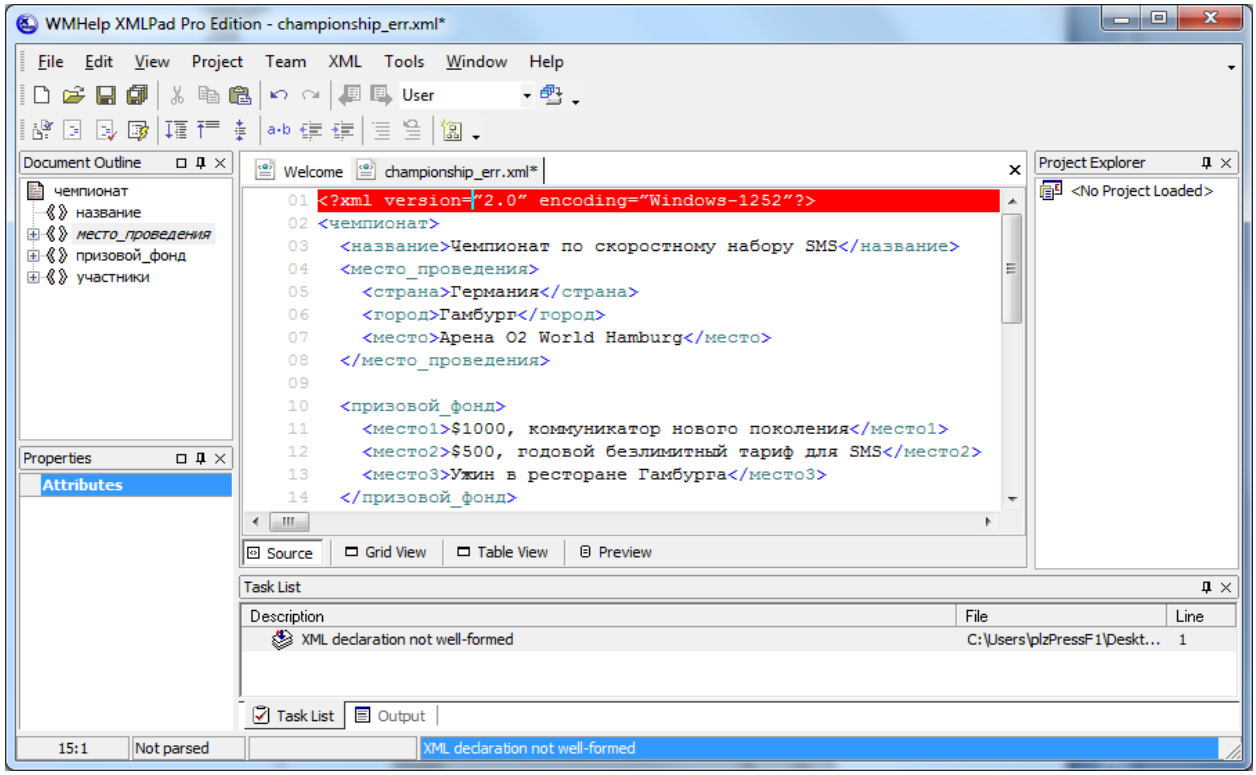


Рисунок 1.12. Главное окно программы, показывающее место ошибки.

Программа сообщает нам о том, что объявление XML-документа некорректно. Дело в том, что в этом объявлении используются символы псевдокавычек (″) вместо символов кавычек (“).

Задание на работу

1) Ознакомьтесь со следующим XML-документом:

```
<?XML version = 1.0 encoding = WINDOWS-1251?>
<DVD жанр = боевик>
<название>Гладиатор</название>
<год знач = "2000">
<страна>США, Великобритания</страна>
<режиссер>Ридли Скотт</Режиссер>
<DVD жанр = фантастика>
<название>Зеленая миля</название>
<год знач = "1999">
<Страна>США</страна>
<режиссер>Фрэнк Дарабонт</режиссер>
</DVD>
```

В этом документе хранится список коллекции фильмов кинолюбителя. Однако при составлении этого документа были допущены ошибки.

2) Найдите и исправьте ошибки в предоставленном документе. Проверьте правильность структуры XML одним из перечисленных выше способов.
3) Дополните список коллекции фильмов так, чтобы он содержал 10-20 наименований. Помимо **DVD** носителей необходимо расширить коллекцию также и носителями **Blue-rayDisc (BD)**.
4) Проверьте правильность построения дополненного документа XML.

4 Содержание отчёта

- Описание цели лабораторной работы
- Предоставление результатов поиска и устранения ошибок
- Предоставление дополненного правильно оформленного документа XML

Лабораторная работа №2
«Действительные документы XML. DTD»

2.1 Цели лабораторной работы

Целью работы является получение сведений о возможных ограничениях для действительных документов XML, способах определения действительной структуры документа, DTD.

2.2 Содержание работы

- Возможные ограничения для действительных документов XML
- DTD
- Задание для самостоятельной работы

2.3 Порядок выполнения работы

Возможные ограничения для действительных документов XML

В предыдущей лабораторной работе Вы познакомились с синтаксическими правилами XML. Узнали, что документы XML бывают *правильно построенными* и *действительными*. Первая группа соответствует всем общим правилам синтаксиса XML. Однако структура – это несколько большее, чем просто синтаксис. Например, документ XML синтаксически может быть безупречным, но содержать элементы в неправильном порядке, или в нём могут отсутствовать элементы, необходимые обработчику для выполнения транзакции электронной коммерции или решения другой задачи. Соответствие документа некоторым внешним правилам делает его *действительным*.

Все эти правила содержания можно объединить в виде схемы, которая определяет структуру экземпляра XML. Разработчик создаёт схему для применения к документу, который предназначен для решения определённой проблемы; кроме того, возможен вариант выбора схемы из заранее созданной общедоступной библиотеки.

Схема чётко определяет имя и структуру корневого элемента, включая спецификацию всех его дочерних элементов. Разработчик может задать, какие элементы и в каком количестве обязательны, а какие – необязательны. Схема определяет, какие элементы содержат атрибуты, а также допустимые значения этих атрибутов, в том числе и значения по умолчанию. Как и в случае элементов, атрибуты могут быть обязательными и необязательными.

DTD

В программировании на XML используется несколько разновидностей схем. Мы рассмотрим схему **DTD** (англ. *DocumentTypeDefinition* определение типа документа).

В XML-документах DTD определяет набор действительных элементов, идентифицирует элементы, которые могут находиться в других элементах, и определяет действительные атрибуты для каждого из них.

В DTD для XML используются следующие типы правил: правила для элементов и их атрибутов, описания категорий (макроопределений), описание форматов бинарных данных. Все они описывают основные конструкции языка – элементы, атрибуты, символьные константы внешние файлы бинарных данных.

Для того, чтобы использовать DTD в нашем документе, мы можем или описать его во внешнем файле и при описании DTD просто указать ссылку на этот файл или же непосредственно внутри самого документа выделить область, в которой определить нужные правила. В первом случае в документе указывается имя файла, содержащего DTD-описания:

```
<?xml version="1.0"?>
<! DOCTYPE championship SYSTEM "championship.dtd">
```

Внутри же документа DTD-декларации включаются следующим образом:

```
...
<! DOCTYPEchampionship [
<!ELEMENT чемпионат (название, место_проведения, призовой_фонд?, участники)>
…
]>
```

В том случае, если используются одновременно внутренние и внешние описания, то программой-анализатором будут сначала рассматриваться внутренние, т.е. их приоритет выше. При проверке документа XML-процессор в первую очередь ищет DTD внутри документа. Если правила внутри документа не определены и не задан атрибут *standalone ="yes"* , то программа загрузит указанный внешний файл и правила, находящиеся в нем, будут считаны оттуда. Если же атрибут *standalone*имеет значение *"yes*", то использование внешних DTD описаний будет запрещено.

Определение элемента

Элемент в DTD определяется с помощью дескриптора **!ELEMENT**, в котором указывается название элемента и структура его содержимого.

Например, для элемента <*страна*> можно определить следующее правило:

```
<!ELEMENT страна (#PCDATA)>
```

Ключевое слово ELEMENT указывает, что данной инструкцией будет описываться элемент XML. Внутри этой инструкции задается название элемента (*страна*) и тип его содержимого.

В определении элемента мы указываем сначала название элемента (*страна*), а затем его модель содержимого – определяем, какие другие элементы или типы данных могут встречаться внутри него. В данном случае содержимое элемента *страна* будет определяться при помощи специального маркера PCDATA (что означает parseablecharacterdata – любая информация, с которой может работать

программа-анализатор). Существует еще две инструкции, определяющие тип содержимого: EMPTY, ANY. Первая указывает на то, что элемент должен быть пустым (например, <red/>), вторая – на то, что содержимое элемента специально не описывается.

Последовательность дочерних для текущего элемента объектов задается в виде списка разделенных запятыми названий элементов. При этом для того, чтобы указать количество повторений включений этих элементов могут использоваться символы +,*, ?:

```
<!ELEMENT книга (название, автор+, содержание?)>
```

В этом примере указывается, что внутри элемента *<книга>* должны быть определены элементы *название*, *автор* и *содержание*, причем элемент *название* является обязательным элементом и может встречаться лишь однажды, элемент *автор* может встречаться несколько раз, а элемент *содержание* является опциональным, т.е. может отсутствовать. В том случае, если существует несколько возможных вариантов содержимого определяемого элемента, их следует разделять при помощи символа "|":

```
<!ELEMENT flower (PCDATA | title )*>
```

Символ * в этом примере указывает на то, что определяемая последовательность внутренних элементов может быть повторена несколько раз или же совсем не использоваться.

Если в определении элемента указывается "смешанное" содержимое, т.е. текстовые данные или набор элементов, то необходимо сначала указать PCDATA, а затем разделенный символом "|" список элементов.

Пример корректного XML- документа:

```
<?xmlversion="1.0"?>
<! DOCTYPE journal [
<!ELEMENT contacts (address, tel+, email?)>
<!ELEMENT address (street, appt)>
<!ELEMENT street PCDATA>
<!ELEMENTappt (PCDATA | EMPTY)*>
<!ELEMENTtel PCDATA>
<!ELEMENT email PCDATA>
]>
<contacts>
<address>
<street>Marks avenue</street>
<appt/>
</address>
<tel>12-12-12</tel>
<tel>46-23-62</tel>
<email>info@j.com</email>
</contacts>
```

Определение атрибутов

Списки атрибутов элемента определяются с помощью ключевого слова **!ATTLIST**. Внутри него задаются названия атрибутов, типы их значений и дополнительные параметры. Например, для элемента <*article*> могут быть определены следующие атрибуты:

```
<!ATTLISTarticle
idID #REQUIRED
about CDATA #IMPLIED
type (actual | review | teach )  'actual' "
>
```

В данном примере для элемента *article*определяются три атрибута: *id, about*и *type*, которые имеют типы ID (идентификатор), CDATA и список возможных значений соответственно. Всего существует шесть возможных типов значений атрибута:

1) CDATA - содержимым документа могут быть любые символьные данные
2) ID - определяет уникальный идентификатор элемента в документе
3) IDREF (IDREFS) – указывает, что значением атрибута должно выступать название (или несколько таких названий, разделенных пробелами во втором случае) уникального идентификатора определенного в этом документе элемента
4) ENTITY (ENTITIES) – значение атрибута должно быть названием (или списком названий, если используется ENTITIES) компонента (макроопределения), определенного в документе
5) NMTOKEN (NMTOKENS) – содержимым элемента может быть только одно отдельное слово (т.е. этот параметр является ограниченным вариантом CDATA)
6) Список допустимых значений – определяется список значений, которые может иметь данный атрибут.

Также в определении атрибута можно использовать следующие параметры:

- #REQUIRED – определяет обязательный атрибут, который должен быть задан во всех элементах данного типа
- #IMPLIED – атрибут не является обязательным
- #FIXED "значение" – указывает, что атрибут должен иметь только указанное значение, однако само определение атрибута не является обязательным, но в процессе разбора его значение в любом случае будет передано программе-анализатору
- Значение – задает значение атрибута по умолчанию

Определение компонентов (макроопределений)

Компонент (entity) представляет собой определения, содержимое которых может быть повторно использовано в документе. В других языках программирования подобные элементы называются макроопределениями. Создаются DTD-компоненты при помощи инструкции !ENTITY:

```
<!ENTITY hello ' Мы рады приветствовать Вас!' >
```

Программа-анализатор, просматривая в первую очередь содержимое области DTD-определений, обработает эту инструкцию и при дальнейшем разборе документа будет использовать содержимое DTD-компонента в том месте, где будет встречаться его название. Т.е. теперь в документе мы можем использовать выражение “&hello;”, которое будет заменено на строчку *"Мы рады приветствовать Вас"*.

В общем случае, внутри DTD можно задать три типа макроопределений:

Внутренние макроопределения – предназначены для определения строковой константы, с их помощью можно организовывать ссылки на часто изменяемую информацию, делая документ более читабельным. Внутренние компоненты включаются в документ при помощи амперсанта &

В XML существует пять предустановленных внутренних символьных констант:

- < - символ "<"
- > - символ ">"
- & - символ "&"
- ' - символ апострофа "'"
- " - символ двойной кавычки """

Внешние макроопределения – указывают на содержимое внешнего файла, причем этим содержимым могут быть как текстовые, так и двоичные данные. В первом случае в месте использования макроса будут вставлены текстовые строки, во втором – бинарные данные, которые анализатором не рассматриваются и используются внешними программами

```
<!ENTITY logotype SYSTEM "/image.gif" NDATA GIF87A>
```

Макроопределения правил – макроопределения параметров могут использоваться только внутри области DTD и обозначаются специальным символом %, вставляемым перед названием макроса. При этом содержимое компонента будет помещено непосредственно в текст DTD-правила

Например, для следующего фрагмента документа

```
<!ELEMENTname (PCDATA)>
<!ELEMENT title (PCDATA | name)*>
<!ELEMENT author (PCDATA | name)*>
<!ELEMENT article (title, author)*>
<!ELEMENT book (title, author)*>
<!ELEMENT bookstore (book | article)*>
<!ELEMENT bookshelf (book | article)*>
```

можно использовать более короткую форму записи:

```
<!ELEMENT name (PCDATA)>
<! ENTITY %names 'PCDATA | name'>
<!ELEMENT article (%names;)*>
```

```
<!ELEMENT book (%names;)*>
<!ENTITY %content 'book | article'>
<!ELEMENT bookstore (%content;)*>
<!ELEMENT bookshelf (%content;)*>
```

Типизация данных

Довольно часто при создании XML-элемента разработчику требуется определить, данные какого типа могут использоваться в качестве его содержимого. Т.е. если мы определяем элемент <last-modified>10.10.98</last-modified>, то хотим быть уверенными, что в документе в этом месте будет находиться строка, представляющая собой дату, а не число или произвольную последовательность символов. Используя типизацию данных, можно создавать элементы, значения которых могут использоваться, например, в качестве параметров SQL-запросов. Программа клиент в этом случае должна знать, к какому типу данных относится текущее значение элемента и в случае соответствия формирует SQL-запрос.

Если в качестве программы на стороне клиента используется верифицирующий XML-процессор, то информацию о типе можно передавать при помощи специально созданного для этого атрибута элемента, имеющего соответствующее DTD-определение. В процессе разбора программа-анализатор передаст значение этого атрибута клиентскому приложению, которое сможет использовать эту информацию должным образом. Например, чтобы указать, что содержимое элемента должно быть длинным целым, можно использовать следующее DTD- определение:

```
<!ELEMENT counter (PCDATA)>
<!ATTLIST counter data_long CDATA #FIXED "LONG">
```

Задав атрибуту значение по умолчанию LONG и определив его как FIXED, мы позволили тем самым программе-клиенту получить необходимую информацию о типе содержимого данного элемента, и теперь она может самостоятельно определить соответствие типа этого содержимого указанному в DTD-определении.

В DTD можно определять следующие основные типы данных:
"CURRENCY", "BYTE", "INTEGER", "FLOAT", "LONG", "BOOL"

В заключении хотелось бы отметить, что DTD предоставляет нам весьма удобный механизм осуществления контроля за содержимым документа. На сегодняшний день, практически все программы просмотра документов Интернет используют DTD-правила. Однако это далеко не единственный способ проверки корректности документа. В настоящий момент статус рекомендации W3 консорциума имеет новый стандарт языка описания структуры документов, называемый схемами данных (XMLSchema).

Задание на работу

1) Определите схему DTD для документа XML из прошлой лабораторной работы, содержащего сведения о коллекции фильмов.

2) Используйте внешнее или внутренне определение на свой выбор.
3) Проверить удовлетворяет ли документ написанным правилам. Для этого можно использовать рассмотренную ранее программу XMLPad. Для этого загрузите документ **xml** в программу и выберите пункт меню «Validate» в меню «XML». В случае, если какой-либо элемент или атрибут не были определены, программа выдаст список несоответствий как показано на рисунке 2.1.

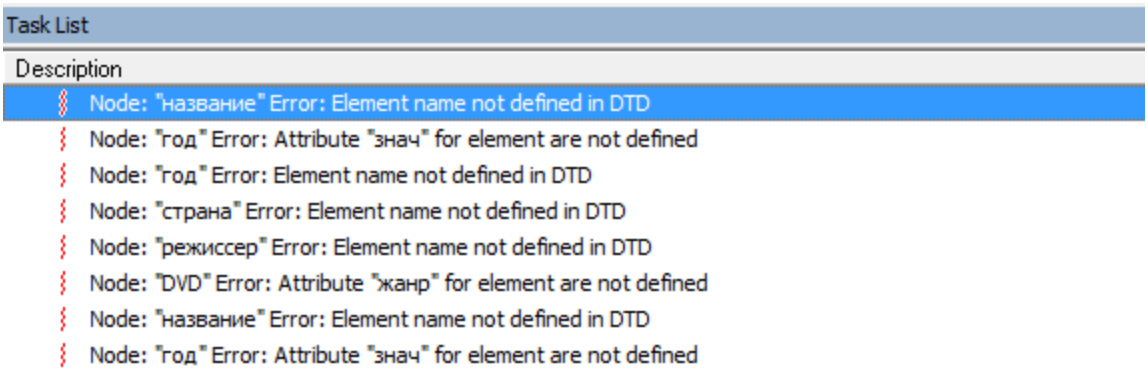


Рисунок 2.1. Список несоответствий заданной схеме.

В случае соответствия документа схеме программа выдаст сообщение, показанное на рисунке 2.2.

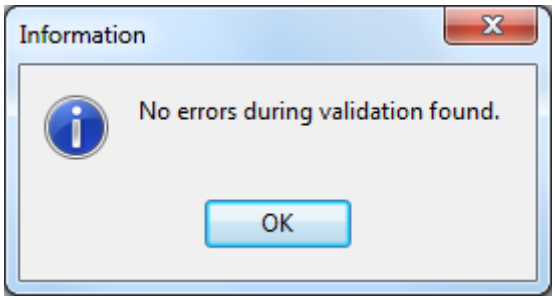


Рисунок 2.2. Сообщение об отсутствии ошибок.

2.4 Содержание отчёта

- Описание сущности действительных документов, понятия *схема документа*
- Предоставление правильно составленной схемы DTD

Лабораторная работа №3 «Общие сведения о XML и XSL. Программа "XSL-процессор версии 2.0"»

1 Цели лабораторной работы

Целью работы является получение общих сведений о XML и XSL, знакомство с программой "XSL-процессор версии 2.0" и выполнение XSL-преобразования при помощи этой программы.

2 Содержание работы

- Общие сведения о XML и XSL
- Знакомство с программой "XSL-процессор версии 2.0"
- Выполнение XSL-преобразования при помощи программы для существующих документов

3 Порядок выполнения работы

Общие сведения о XML и XSL

XML (*ExtensibleMarkupLanguage*) – это язык разметки, описывающий целый класс объектов данных, называемых XML-документами. Этот язык используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов. Т.е. сам по себе XML не содержит никаких тэгов, предназначенных для разметки, он просто определяет порядок их создания.

Технология XML широко используется в интернете, а также для передачи документов между приложениями. С помощью XML можно создавать свою собственную структуру данных.

Простейший пример:

```
<?xmlversion="1.0" encoding="WINDOWS-1251"?>
<tutorial>
<title>"Заметкиоб XML"</title>
</tutorial>
```

Здесь мы видим два тега <tutorial> и <title>. Оба тега имеют закрывающую пару </tutorial> и </title>. Первая строка – объявление, что файл является xml документом. Из этого сразу видно, что XML похож на известный стандарт HTML. Ключевая разница в том, что теги можно придумывать самим и записать и структурировать в виде DTDдокумента (файла). Можно придумать свою собственную структуру, а также определять типы данных и дополнительные атрибуты.

Пример простого книжного магазина.

Пусть в нашем магазине есть категории товаров (жанры). Каждая книга принадлежит какому-то жанру и имеет информацию об авторе, названии и цене.

ПустьXMLсодержитданные:

```
<?xmlversion="1.0" encoding="windows-1251"?>
```

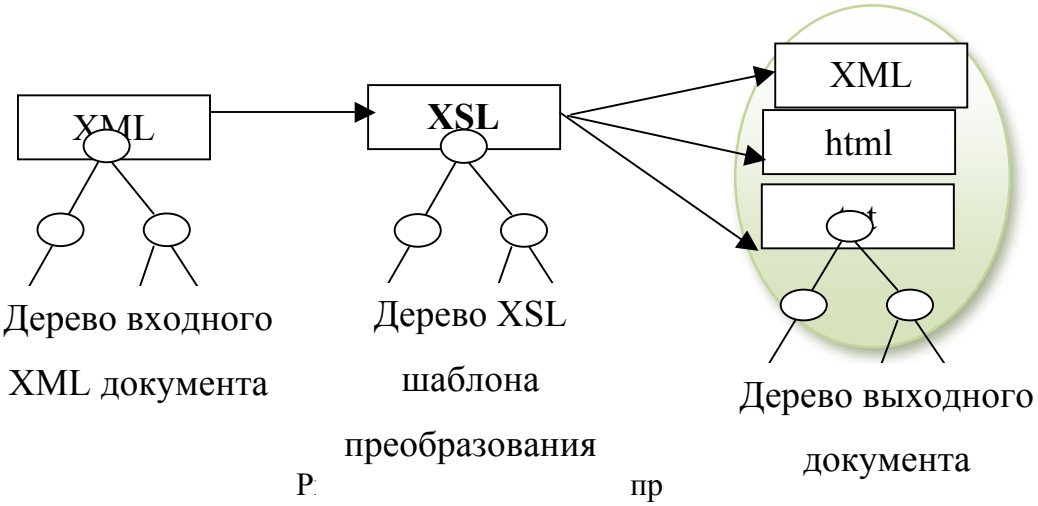
```
<книги>
<книга категория="справочник">
<автор>Найджел Рис</автор>
<название>Поговорки века</название>
<цена>90</цена>
</книга>
<книга категория="фантастика">
<автор>Сергей Лукьяненко</автор>
<название>Лабиринт Отражений</название>
<цена>150</цена>
</книга>
<книга категория="детектив">
<автор>Агата Кристи</автор>
<название>12 негритят</название>
<цена>120</цена>
</книга>
<книга категория="роман">
<автор>Чарльз Диккенс</автор>
<название>Большие надежды</название>
<цена>200</цена>
</книга>
<книга категория="фентези">
<автор>Дж. Р. Р. Толкиен</автор>
<название>Хоббит</название>
<цена>300</цена>
</книга>
<книга категория="фантастика">
<автор>Курт Воннегут</автор>
<название>Бойня №5</название>
<цена>140</цена>
</книга>
</книги>
```

Данный XML есть возможность просматривать прямо в браузере. В зависимости типа и версии браузера вывод будет иметь различную форму, что не всегда удобно.

Поэтому появилась ещё одна смежная с XML технология – XSL процессор.

XSL (eXtensibleStylesheetLanguage) — это язык описания стилей отображения для XML-документов. Он позволяет нам описать способ отображения документов для различных носителей, будь то окно Веб-обозревателя, устройство печати или генератор речи. XSL-процессор принимает данные в формате XML и таблицу стилей в формате XSL и на их основе создает отображение данных в соответствии с заданными стилями. Процесс отображения данных состоит из двух этапов. Сначала строится результирующее дерево на базе исходного дерева XML-документа (этот процесс назвается *преобразованием*), а затем это дерево преобразуется в видимый или слышимый результат (этот процесс называется *форматированием*).

На рисунке 3. 1 представлен механизм преобразования XSL.



Р пр

Само преобразование выполняет XSL процессор на основании XSL файла. Для этого он должен понимать структуру входного XML файла, поэтому первым делом он строит дерево входного XML файла. Сам файл XSL, как будет показано ниже, сам имеет структуру дерева. Руководствуясь описанными в нём правилами, на выходе получается файл новой структуры, который также имеет новую структуру. Таким образом, для XSL преобразования необходимо построить 3 дерева. Как правило, построение деревьев и само преобразование берёт на себя XSL процессор.

Знакомство с программой ”XSL-процессор версии 2.0”

Назначение программы.

Программа предназначена для удобного и наглядного трансформирования и реструктурирования исходных XML файлов в выходные файлы расширений XML, HTML и TXT посредством применения XSL шаблонов стилей и соответствующего бесплатного XSLтрансформераMSXSL.

Возможности программы.

Программа может работать в двух режимах – в ручном и демонстрационном.

В ручном режиме предусмотрена возможность внешнего импорта исходных файлов, а в демонстрационном режиме предусмотрен выбор исходных файлов из списка. Также в программе предусмотрены возможность сохранения результата трансформации документа и возможность увидеть результаты каждого основного этапа работы программы.

Внешний интерфейс программы.

Внешний вид программы представлен на рисунке 3.2.

Области по номерам:

1 – область для загрузки исходных файлов в ручном режиме;
2 – область для выбора исходных файлов в демонстрационном режиме;
3 – область для переключения режимов работы программы;
4 – область для управления работой программы;

5 – область отображения работы программы.

Область 1. При выбранном ручном режиме она имеет зелёный оттенок, в противном случае имеет обычный серый оттенок. Содержит две кнопки – открыть файл XML и открыть файл XSL шаблона стилей. Диалоговое окно выбора файла изображено на рисунке 3.3

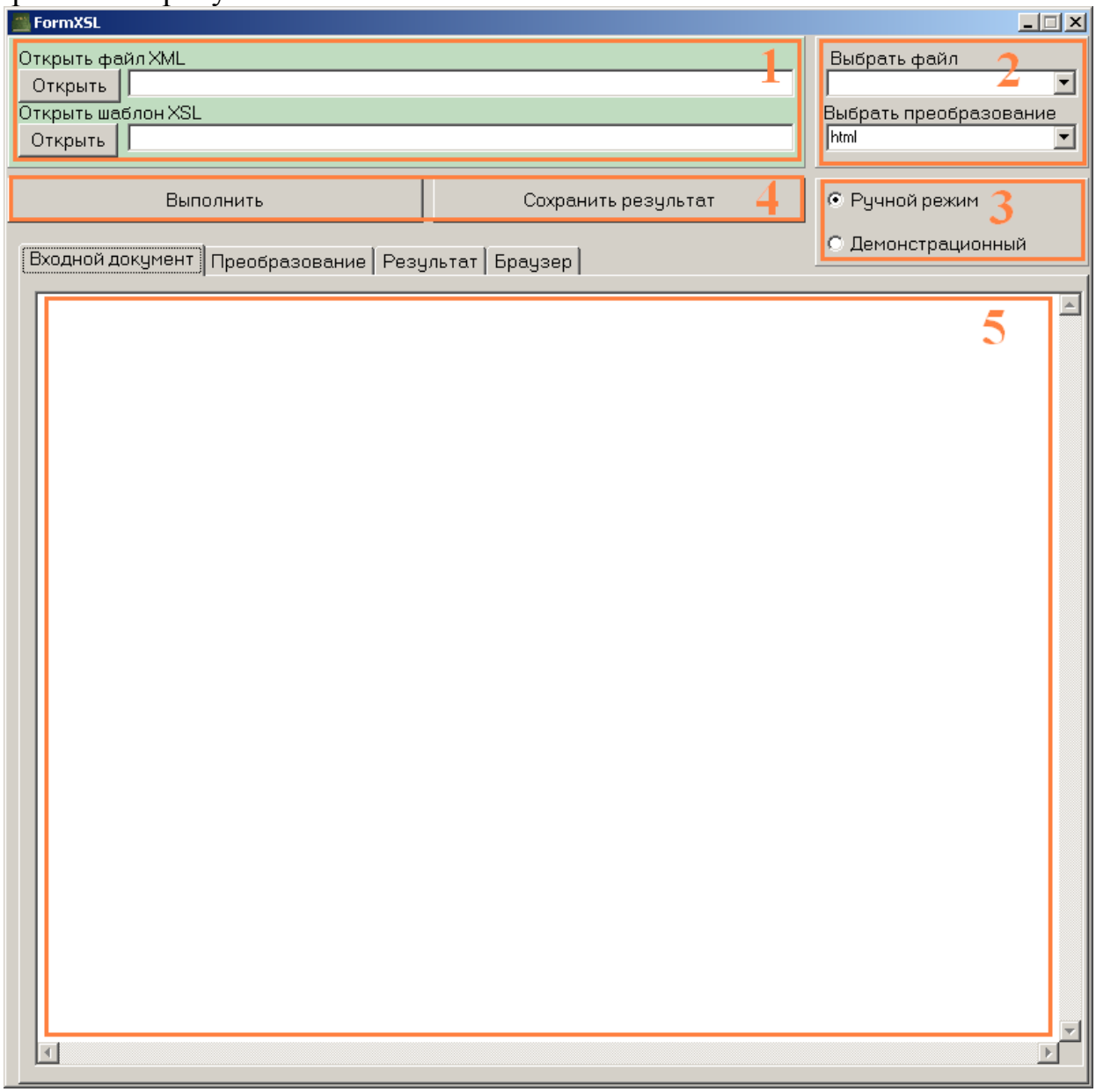


Рисунок.3.2. Внешний вид программы.

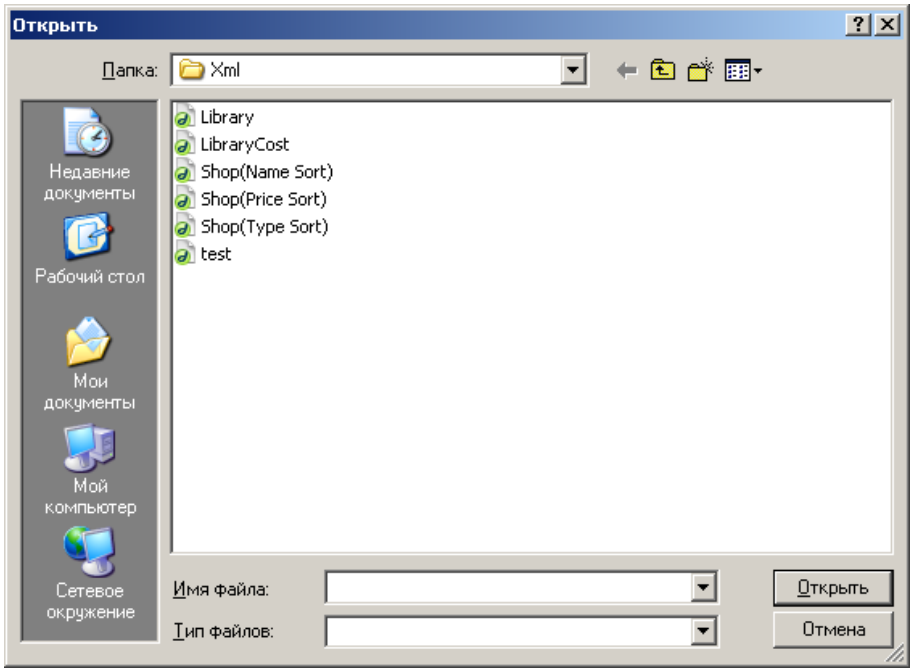


Рисунок3.3Диалоговое окно выбора файла.

После выбора желаемого файла его имя и полный путь к нему записывается в текстовое поле правее от кнопки. В случае отмены выбора файла текстовое поле остаётся пустым. По умолчанию предлагается выбрать файл по адресу:

[путь к папке программы] ->Docs –>Xml

Область 2. При выбранном демонстрационном режиме она имеет зелёный оттенок, в противном случае имеет обычный серый оттенок. Содержит две выпадающие кнопки списка: выбрать файл XML и выбрать преобразование. В этом случае для выбора XML будут доступны файлы, размещённые по адресу:

[путь к папке программы] ->Docs –>Xml

Выбор преобразования всегда имеет три значения – XML, HTML и TXT. При этом важно, чтобы файлы преобразований назвались так же, как и исходный XML файл, и располагались в соответствующих папках. К примеру, если выбран файл по адресу

[путь к папке программы] ->Docs –>Xml –>“Library.xml”,

то его шаблоны должны располагаться по адресам:

для txt преобразования

[путь к папке программы] ->Docs –>Xsl ->TXT -> “ Library.xsl”

для xml преобразования

[путь к папке программы] ->Docs –>Xsl ->Xml -> “ Library.xsl”

для html преобразования

[путь к папке программы] ->Docs –>Xsl ->HTML -> “ Library.xsl”

В случае выбора преобразования по несуществующему шаблону возникнет ошибка.

Область 3. Предназначена для смены режимов ручной/демонстрационный. Для выбранного режима его область подсвечивается бледно зелёным цветом, а область другого режима становится серой и недоступная для изменения.
Область 4. Имеет две кнопки. Копка выполнить – совершает искомое преобразование и записывает все шаги работы программы в окна области 5. Кнопка сохранить позволяет сохранить результат преобразования в выходном файле. В демонстрационном режиме расширение сохраняемого фала может быть вписано автоматически. Сохранение файла производится по умолчанию по адресу
[путь к папке программы] ->Docs –>Out ->[Имя файла и расширение]
Окно диалога сохранения файла изображено на рисунке 3.4.

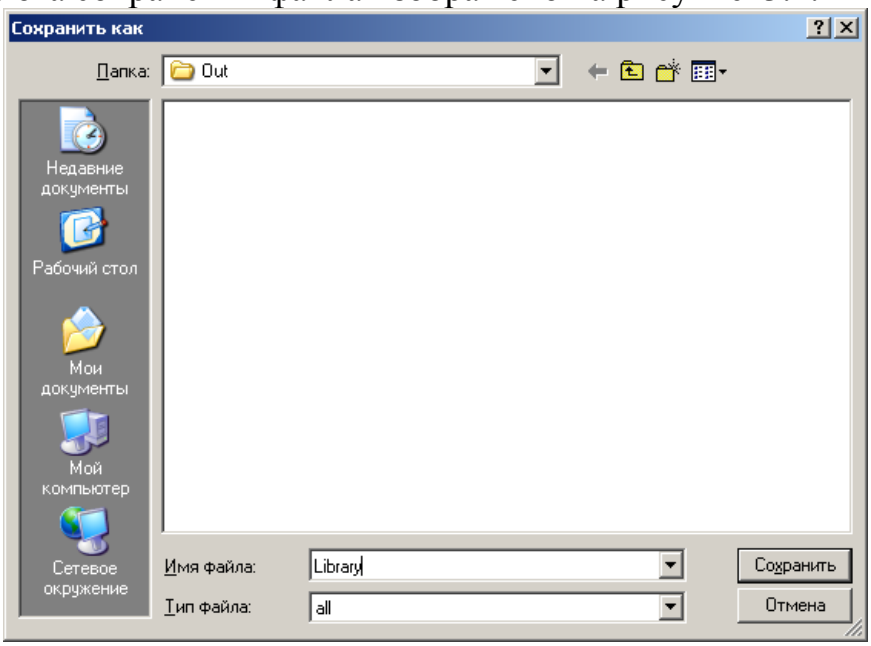


Рисунок 3.4. Диалоговое окно сохранения файла.

В случае возникновения ошибок выбора входных файлов, их содержания и хода преобразования предусмотрен вывод сообщения о соответствующей ошибке в новом окне поверх окна программы.

Область 5. Предназначена для отображения хода работы программы. Четыре вкладки позволяют наглядно проследить всю цепочку преобразований:

- вкладка “Входной документ” – здесь выводится загружаемый xml файл;
- вкладка “Преобразование” – здесь выводится файл шаблона XSL;
- вкладка “Результат” – здесь выводится конечный файл, получаемый из исходного, путём преобразования в соответствии с выбранным XSL стилем;
- вкладка “Браузер” – здесь выводится результат, как бы он отображался в браузере. Этой вкладкой имеет смысл пользоваться, когда на выходе нужно получить html файл.

Результат преобразования после каждой удачной трансформации сохраняется в буферный файл “Buf.html” в папке с исполняемым файлом программы. Наличие этого файла обусловлено механикой работы окна браузера, которое может отображать только цельные файлы (а не содержание памяти).

Выполнение XSL-преобразования при помощи программы для существующих документов

В качестве демонстрационных файлов к программе прилагается набор файлов – модель книжного магазина.

Файлы модели книжного магазина расположены по адресам:

1) [путь к папке программы] ->Docs –>Xml ->”Library.xml” – основной исходный файл перечня книжных изделий;
2) [путь к папке программы] ->Docs –>Xml ->”LibraryCost.xml” – вспомогательный исходный файл, содержащий информацию о стоимости книжных изделий по категориям;
3) [путь к папке программы] ->Docs –>Xsl–>html ->”Library.xsl” – основной шаблон стилей, совершающий основное преобразование, - отображение отсортированного в алфавитном порядке списка книжных изделий по всем имеющимся жанрам;
4) [путь к папке программы] ->Docs –>Xsl–>html ->” LibraryCost.xsl” – вспомогательный шаблон стилей, совершающий пересчёт цены книжных изделий в соответствии с указанной категорией;
5) [путь к папке программы] ->Docs –>Xsl–>xsl ->”Library.xsl” – шаблон стилей для отображения библиотеки книжных изделий в другом по структуре Xml файле;
6) [путь к папке программы] ->Docs –>Xsl–>txt ->”Library.xsl” – шаблон стилей для отображения библиотеки книжных изделий в текстовом виде (для вывода на печать).

Модель библиотеки предназначена для отображения главным образом в демонстрационном режиме. Поэтому, при выборе соответствующего html преобразования главная форма программы приобретает вид как на рисунке 3.5.
При этом появляются новые 4 вкладки с именами отображаемых файлов. Схема работы преобразования представлена на рисунке 3.6.

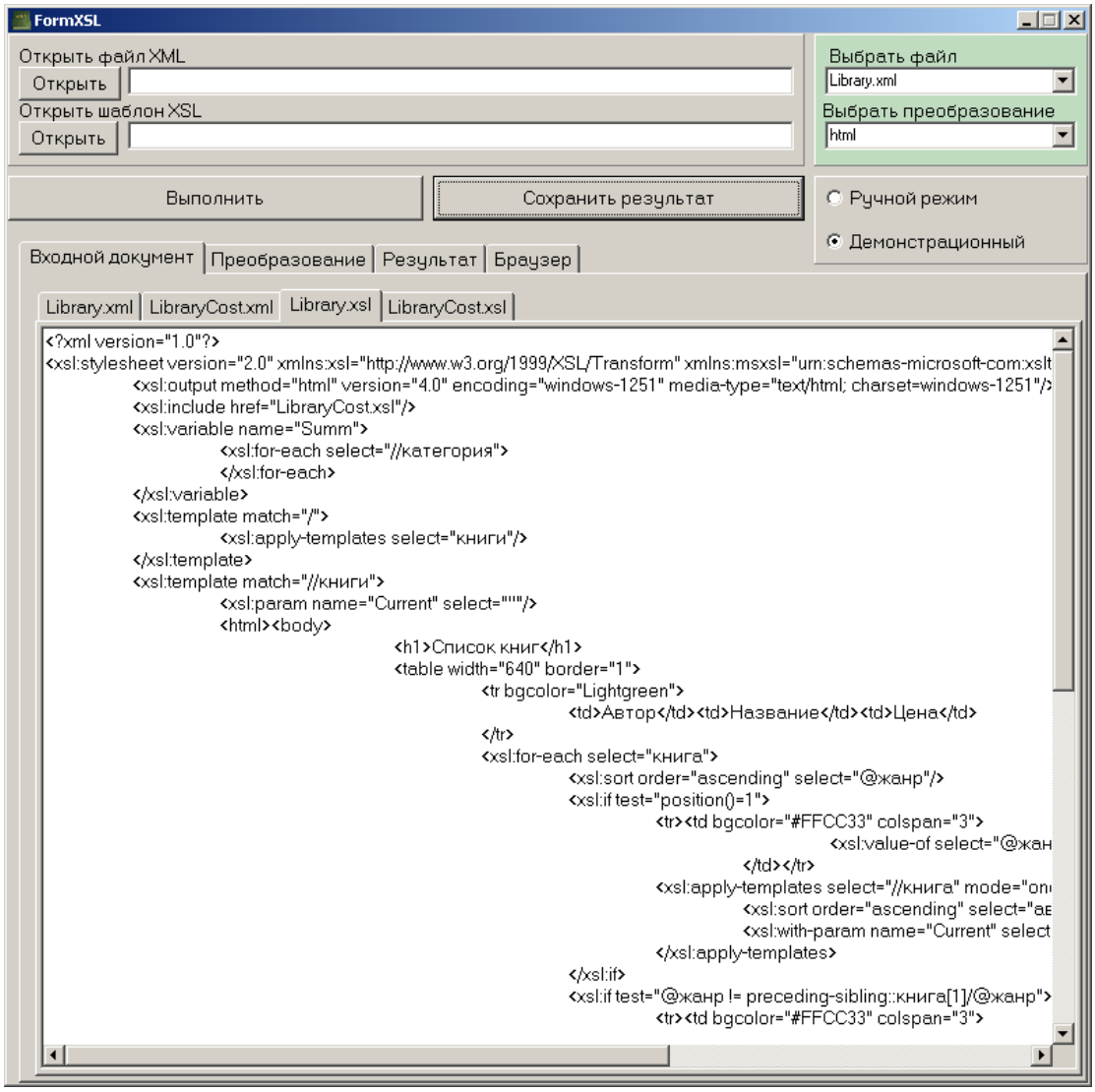


Рисунок 3.5. Внешний вид программы при совершении html преобразования модели книжного магазина в демонстрационном режиме.

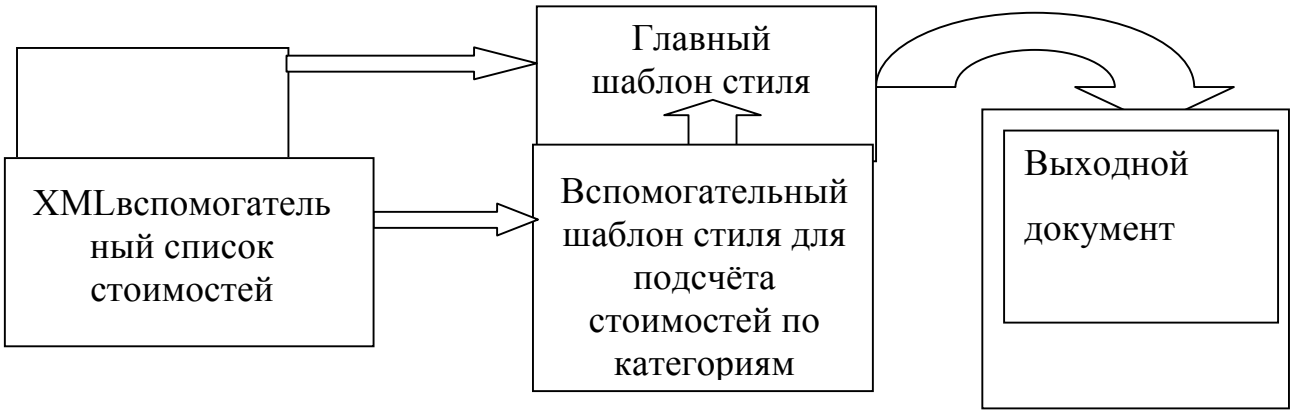


Рисунок 3.6. Структура модели книжного магазина.

Данная модель демонстрирует, как можно использовать подключение вспомогательных таблиц стилей, а также как обрабатывать сразу несколько

исходных XSL файлов. Помимо этого, показаны возможности языка XPath, такие, как сортировка в алфавитном порядке, циклический вызов шаблонов, подсчёт новых значений стоимостей и работа с переменными. На рисунке 3.7 представлен результирующий HTML файл отчёта. А на рисунке 3.8 представлено отображение результата как бы он выглядел в браузере.

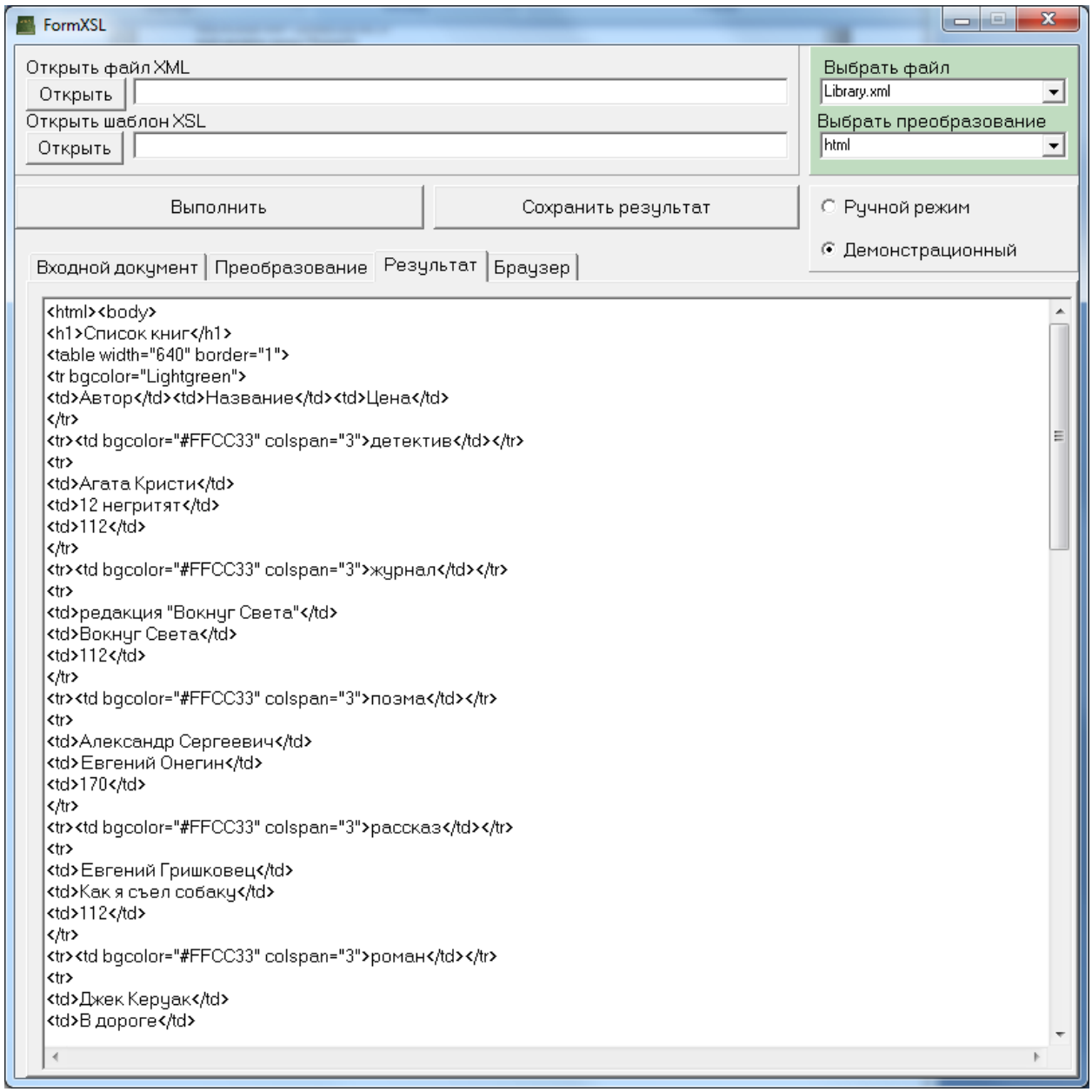


Рисунок 3.7. Результирующий файл формата html.

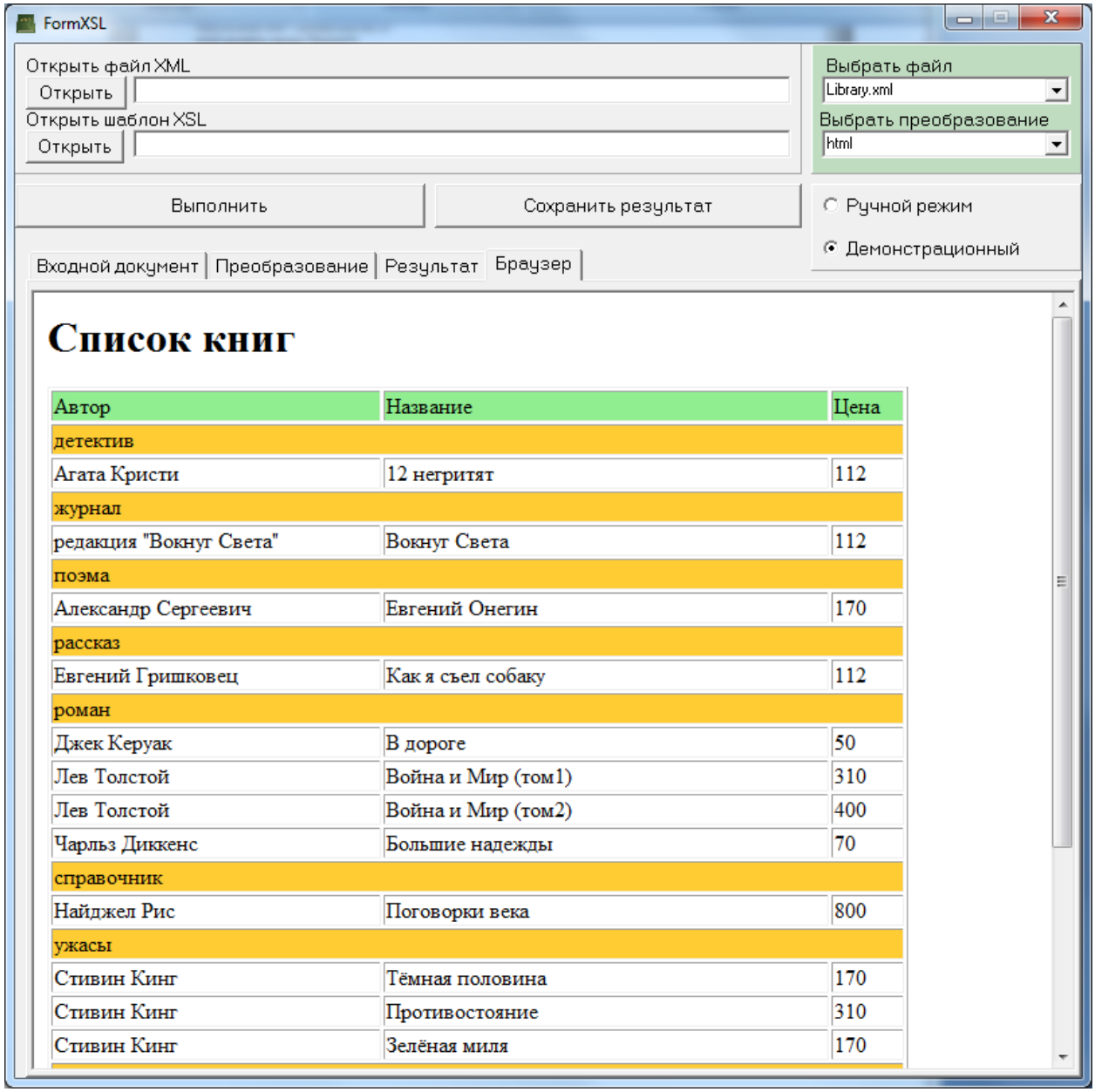


Рисунок 3.8. Отображение результата в браузере.

В последующих лабораторных работах Вам предстоит работать с программой ”XSL-процессор версии 2.0” в ручном режиме. Для этого в третьей области необходимо выбрать соответствующий пункт, а затем в первой области выбрать файл XML и соответствующее для него преобразование (файл-шаблон XSL). В рамках этой лабораторной работы необходимо выполнить XSL-трансформацию в ручном режиме для заранее подготовленных файлов. В качестве исходных XML-документов предоставлены следующие файлы:

[путь к папке программы] ->Docs –>Xml ->”Shop(NameSort).xml”
[путь к папке программы] ->Docs –>Xml ->”Shop(PriceSort).xml”
[путь к папке программы] ->Docs –>Xml ->”Shop(TypeSort).xml”

В них находится список товаров компьютерного магазина, а также указан критерий, по которому будет производиться сортировка во время XSL-трансформации - по имени, порядковому номеру и типу комплектующих соответственно.

В качестве XSL-шаблонов доступны следующие файлы:

[путь к папке программы] ->Docs –>Xsl ->”ShopAll.xsl”
[путь к папке программы] ->Docs –>Xsl ->”ShopCards.xsl”
[путь к папке программы] ->Docs –>Xsl ->”ShopHard.xsl”
[путь к папке программы] ->Docs –>Xsl ->”ShopIO.xsl”

Они необходимы для отображения комплектующих в формате **html**. При этом первый шаблон выводит на экран все комплектующие. В то время как остальные три шаблона – карты памяти, комплектующие начинки системного блока и устройства ввода-вывода соответственно.

К любому из исходных XML-документов можно применять любой шаблон. Выполните XSL-трансформацию в ручном режиме для всех исходных XML-документов, применив к ним как минимум по два различных шаблона.

4 Содержание отчёта

- Описание цели лабораторной работы и сущности понятий XML и XSL
- Предоставление результатов XSL-преобразования для ручного режима

Лабораторная работа №4
«Работа с шаблонами XSL»

Цели лабораторной работы

Целью работы является получение общих сведений о принципах работы XSLT-процессора, выполнение простейших преобразований исходного документа для превращения в другие форматы (**txt**, **html**), просмотр результатов преобразований при помощи программы ”XSL-процессор версии 2.0”.

Содержание работы

- Описание принципов работы XSLT-процессора
- Примеры преобразований документа XML в формат **html**.
- Задание для самостоятельной работы с языком XSLT

Порядок выполнения работы

Описание принципов работы XSLT-процессора

XSLT (*e**X**tensible**S**tylesheet**L**anguage**T**ransformations*) — часть спецификации XSL, задающая язык преобразований XML-документов. Спецификация XSLT является рекомендацией W3C.

При применении *таблицы стилей* XSLT, состоящей из набора *шаблонов*, к XML-документу (*исходное дерево*) образуется *конечное дерево*, которое может быть другой XML-структурой, HTML-документом или обычным текстом. Правила выбора (и, отчасти, преобразования) данных из исходного дерева пишутся на языке запросов XPath.

XSLT имеет множество различных применений, в основном в области web-программирования и генерации отчётов. Одной из задач, решаемых языком XSLT, является отделение данных от их представления, как часть общей парадигмы MVC (англ. *Model-view-controller*). Другой стандартной задачей является преобразование XML-документов из одной XML-схемы в другую.

Консорциум W3 определяет три составные части языка XSL (англ. *eXtensibleStylesheetLanguage* — Расширяемый Язык Стилей): XSLT, XPath (язык путей и выражений, используемый в XSLT для доступа к отдельным частям XML-документа) и XSL-FO (англ. *eXtensibleMarkupLanguageFormattingObjects*) — язык разметки типографских макетов и иных предпечатных материалов.

С XSL вы можете свободно модифицировать исходный текст. Так с помощью различных преобразований можно получить различный результат из одного исходного файла. Каждое преобразование XSL должно начинаться с элемента `xsl:stylesheet`. Атрибут `version='1.0'` определяет версию спецификации XSL. Если преобразование не содержит чего-либо помимо заголовка, то используется обработка по умолчанию. XSLT-процессоры анализируют исходный XML и пытаются найти подходящий XSL-шаблон. Если такой шаблон найден, то выполняются инструкции внутри него. Части XML-документа, к которым должен

применяться шаблон, определяются путями адресации. Их синтаксис описан в Спецификации XPath. В простейших случаях он очень похож на адресацию файловой системы. Обработка всегда начинается с шаблона, где `match="/"`. Это значение пути адресации соответствует корневому узлу. Однако многие преобразования XSL не содержат такой шаблон явно. В этом случае используется неявный шаблон (он содержит только единственную инструкцию). Эта инструкция обозначает: обрабатывать все дочерние элементы текущего узла, включая текстовые узлы. Когда шаблон для узла существует, никакой обработки по умолчанию не происходит. Если вы хотите включить обработку потомков узла, то вы должны явно указать шаблоны для них. Соответствие шаблонов определяется путями адресации, где конкретные пути разделяются с помощью символа "|". Символ "*" обозначает все возможности. С помощью режимов (modes) элемент может быть обработан многократно, причем каждый раз с различным результатом.
Достаточно часто несколько шаблонов соответствует одному и тому же элементу в исходном XML. Поэтому надо решить, какой из них следует использовать. Для этого можно определить приоритеты с помощью атрибута `priority`. Если этот атрибут не определен, его приоритет вычисляется в соответствии с несколькими правилами. Может случиться так, что несколько шаблонов имеют одинаковый приоритет. В таком случае XSLT-процессор может сообщить об ошибке. Если этого не произошло, XSLT-процессор должен выбрать среди соответствующих шаблонов тот, который будет последним в преобразовании. Вычисленные приоритеты располагаются в диапазоне от -0.5 до 0.5. Более подробная информация содержится в Спецификации XSLT.

К атрибутам можно обращаться так же, как и к элементам. Надо только поставить "@" перед именем атрибута. Атрибуты можно обрабатывать аналогично элементам.

Примеры преобразований документа XML в формат *html*

Рассмотрим простейшие примеры преобразований по указанным выше принципам.

- **Пример 1. Использование различных шаблонов.**

Исходный XML

```
<source>
<bold>Hello, world.</bold>
<red>I am </red>
<italic>fine.</italic>
</source>
```

Преобразование XSLT

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

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

<xsl:template match="red">
      <pstyle="color:red">
            <xsl:value-of select="."/>
      </p>
</xsl:template>

<xsl:template match="italic">
      <p>
            <i>
                  <xsl:value-of select="."/>
            </i>
      </p>
</xsl:template>
</xsl:stylesheet>
```

Результат

```
<p>
  <b>Hello, world.</b>
</p>
<p style="color:red">I am </p>
<p>
  <i>fine.</i>
</p>
```

Представление HTML

Hello, world.
I am
fine.

- **Пример 2. Использование символа "|" в путях адресации.**

Исходный XML

```
<source>
<employee>
      <firstName>Joe</firstName>
      <surname>Smith</surname>
</employee>
</source>
```

Преобразование XSLT

```
<xsl:stylesheet version = '1.0'
     xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="firstName|surname">
     <div>
          <xsl:text>[template: </xsl:text>
          <xsl:value-of select="name()"/>
          <xsl:text> outputs </xsl:text>
          <xsl:apply-templates/>
          <xsl:text> ]</xsl:text>
     </div>
</xsl:template>
</xsl:stylesheet>
```

Результат

```
<div>[template: firstName outputs Joe ]</div>
<div>[template: surname outputs Smith ]</div>
```

Представление HTML

[template: firstName outputs Joe]
[template: surname outputs Smith]

- **Пример 3. Использование режимов (modes).**

Исходный XML

```
<source>
<AAA id="a1" pos="start">
     <BBB id="b1"/>
     <BBB id="b2"/>
</AAA>
<AAA id="a2">
     <BBB id="b3"/>
     <BBB id="b4"/>
     <CCC id="c1">
          <CCC id="c2"/>
     </CCC>
     <BBB id="b5">
          <CCC id="c3"/>
     </BBB>
</AAA>
</source>
```

Преобразование XSLT

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

```
     <xsl:apply-templates select="//CCC" mode="red"/>
     <xsl:apply-templates select="//CCC" mode="blue"/>
     <xsl:apply-templates select="//CCC"/>
</xsl:template>
<xsl:template match="CCC" mode="red">
     <divstyle="color:red">
          <xsl:value-of select="name()"/><xsl:text> id=</xsl:text>
          <xsl:value-of select="@id"/>
     </div>
</xsl:template>
<xsl:template match="CCC" mode="blue">
     <divstyle="color:blue">
          <xsl:value-of select="name()"/><xsl:text> id=</xsl:text>
          <xsl:value-of select="@id"/>
     </div>
</xsl:template>
<xsl:template match="CCC">
     <divstyle="color:purple">
          <xsl:value-of select="name()"/><xsl:text> id=</xsl:text>
          <xsl:value-of select="@id"/>
     </div>
</xsl:template>
</xsl:stylesheet>
```

Результат

```
<div style="color:red">CCC id=c1</div>
<div style="color:red">CCC id=c2</div>
<div style="color:red">CCC id=c3</div>
<div style="color:blue">CCC id=c1</div>
<div style="color:blue">CCC id=c2</div>
<div style="color:blue">CCC id=c3</div>
<div style="color:purple">CCC id=c1</div>
<div style="color:purple">CCC id=c2</div>
<div style="color:purple">CCC id=c3</div>
```

Представление HTML

CCCid=c1
CCCid=c2
CCC id=c3
CCC id=c1
CCC id=c2
CCC id=c3
CCC id=c1
CCC id=c2
CCC id=c3

- **Пример 4. Использование приоритетов (priority).**

Исходный XML

```
<source>
<AAA id="a1" pos="start">
     <BBB id="b1"/>
     <BBB id="b2"/>
</AAA>
<AAA id="a2">
     <BBB id="b3"/>
     <BBB id="b4"/>
     <CCC id="c1">
          <CCC id="c2"/>
     </CCC>
     <BBB id="b5">
          <CCC id="c3"/>
     </BBB>
</AAA>
</source>
```

Преобразование XSLT

```
<xsl:stylesheet version = '1.0'
     xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="/">
     <xsl:apply-templates select="//CCC"/>
</xsl:template>

<xsl:template match="CCC" priority="3">
     <h3style="color:blue">
          <xsl:value-of select="name()"/>
          <xsl:text> (id=</xsl:text>
          <xsl:value-of select="@id"/>
          <xsl:text>)</xsl:text>
     </h3>
</xsl:template>
<xsl:template match="CCC/CCC" priority="4">
     <h2style="color:red">
          <xsl:value-of select="name()"/>
          <xsl:text> (id=</xsl:text>
          <xsl:value-of select="@id"/>
          <xsl:text>)</xsl:text>
     </h2>
</xsl:template>
</xsl:stylesheet>
```

Результат

```
<h3 style="color:blue">CCC (id=c1)</h3>
<h2 style="color:red">CCC (id=c2)</h2>
<h3 style="color:blue">CCC (id=c3)</h3>
```

Представление HTML

CCC (id=c1)
CCC (id=c2)
CCC (id=c3)

- **Пример 5. Обращение к атрибутам.**

Исходный XML

```
<source>
<dog name="Joe">
     <data weight="18 kg" color="black"/>
</dog>
</source>
```

Преобразование XSLT

```
<xsl:stylesheet version = '1.0'
     xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="dog">
     <p>
          <b>
               <xsl:text>Dog: </xsl:text>
          </b>
          <xsl:value-of select="@name"/>
     </p>
     <p>
          <b>
               <xsl:text>Color: </xsl:text>
          </b>
          <xsl:value-of select="data/@color"/>
     </p>
</xsl:template>
</xsl:stylesheet>
```

Результат

```
<p>
  <b>Dog: </b>Joe</p>
<p>
  <b>Color: </b>black</p>
```

Представление HTML

Dog: Joe
Color: black

Задание по работе

1) Создайте соответствующие файлы для примеров выше и выполните преобразование при помощи программы ”XSL-процессор версии 2.0”.
2) Создайте собственные шаблоны XSL. В качестве исходного файла используйте дополненный документ, содержащий список коллекции фильмов, из предыдущей лабораторной работы. Созданные шаблоны должны выполнять следующие действия:

a) Преобразование исходного документа в формат **html**

В документе должно быть две таблицы, содержащие информацию о фильмах в формате *Название/Режиссёр/Страна/Жанр/Год*. Первая таблица должна содержать данные о фильмах из DVD-коллекции, вторая – для коллекции Blue-ray.

Возможный внешний вид документа представлен на рисунке 1.

DVD

Название	Режиссёр	Страна	Жанр	Год
Гладиатор	Ридли Скотт	США, Великобритания	боевик	2000
Зеленая миля	Фрэнк Дарабонт	США	фантастика	1999
Иван Васильевич меняет профессию	Леонид Гайдай	СССР	комедия	1973
Операция «Ы» и другие приключения Шурика	Леонид Гайдай	СССР	комедия	1965
Престиж	Кристофер Нолан	США, Великобритания	триллер	2006
Форрест Гамп	Роберт Земекис	США	драма	1994

Blue-ray

Название	Режиссёр	Страна	Жанр	Год
Леон	Люк Бессон	Франция	триллер	1994
Шерлок Холмс	Гай Ричи	США, Германия	триллер	2009
Как приручить дракона	Дин ДеБлуа, Крис Сандерс	США	мультфильм	2010
Начало	Кристофер Нолан	США, Великобритания	фантастика	2010

Рисунок 4.1. Вид **html** документа после преобразования.

b) Преобразование исходного документа в формат **txt**

Документ должен быть представлен в следующем виде:

Коллекция содержит следующие DVD:
Фильм «*[Название]*» жанра *[жанр]*, снятый в *[год]* году
Режиссёр фильма: *[Режиссёр]*
Страна: *[Страна]*

[следующий фильм на DVD]

Коллекция содержит следующиеBD:
Фильм «*[Название]*» жанра *[жанр]*, снятый в *[год]* году
Режиссёр фильма: *[Режиссёр]*
Страна: *[Страна]*

[следующий фильм на BD]

На рисунке 4.2 показан вид документа после преобразования.

```
Коллекция содержит следующие DVD:

Фильм "Гладиатор" жанра боевик, снятый в 2000 году.
Режиссёр фильма: Ридли Скотт
Страна: США, Великобритания
------------------------------

Фильм "Зеленая миля" жанра фантастика, снятый в 1999 году.
Режиссёр фильма: Фрэнк Дарабонт
Страна: США
------------------------------

Фильм "Иван Васильевич меняет профессию" жанра комедия, снятый в 1973 году.
Режиссёр фильма: Леонид Гайдай
Страна: СССР
------------------------------

Фильм "Операция «Ы» и другие приключения Шурика" жанра комедия, снятый в 1965 году.
Режиссёр фильма: Леонид Гайдай
Страна: СССР
------------------------------

Коллекция содержит следующие BD:

Фильм "Леон" жанра триллер, снятый в 1994 году.
Режиссёр фильма: Люк Бессон
Страна: Франция
------------------------------

Фильм "Шерлок Холмс" жанра триллер, снятый в 2009 году.
Режиссёр фильма: Гай Ричи
Страна: США, Германия
------------------------------

Фильм "Как приручить дракона" жанра мультфильм, снятый в 2010 году.
Режиссёр фильма: Дин ДеБлуа, Крис Сандерс
Страна: США
------------------------------
```

Рисунок 4.2. Вид **txt** документа после преобразования.

Содержание отчёта

- Описание цели лабораторной работыипринципов работы XSLT-процессора
- Предоставление правильно построенных шаблонов, выполняющих необходимые преобразования

Лабораторная работа №5«Продвинутое использование XSLT. Повторение и сортировка. Условная обработка. Численные вычисления. Строковые функции»

1 Цели лабораторной работы

Целью работы является подробное ознакомление с возможностями XSLT, выполнение более сложных преобразований, а также создание документа XML со структурой, отличной от структуры исходного документа.

2 Содержание работы

- Описание дополнительных возможностей XSLT
- Примеры преобразований
- Задание для самостоятельной работы по преобразованию XML в HTML
- Задание для самостоятельной работы по преобразованию структурыXML

3 Порядок выполнения работы

Описание дополнительных возможностей XSLT

В ходе выполнения предыдущей лабораторной работы Вы уже ознакомились с основными возможностями XSLT. Однако, как и в случае со многими другими языками программирования, XSLT предлагает элементы, используемые для проверки выполнения условий, циклического перебора узлов, сортировки, работы с численными и строковыми выражениями, создания элементов и атрибутов и многое другое.

Инструкция `xsl:for-each` определяет шаблон, который применяется для каждого узла, выбранного с помощью атрибута `select`. Узлы, выбранные с помощью `xsl:for-each` или `xsl:apply-templates` можно отсортировать. Порядок сортировки определяется атрибутом `order`, который может принимать значения *ascending* и `descending`для сортировки по возрастанию и по убыванию соответственно. Сортировка может проводиться в текстовом и числовом режимах; за это отвечает атрибут *data-type*, принимающий значения *text* для текстового режима (используется по умолчанию) и number для числового. Атрибут *case-order*, принимающий значения *upper-first* и *lower-first* определяет порядок следования заглавных и прописных букв для текстового режима.

С помощью `xsl:element` в процессе обработки можно создавать новые элементы. Инструкция `xsl:attribute` также служит для генерирования элементов в процессе обработки. Она предназначена для создания атрибутов элемента, в который она заключена.

Инструкция `xsl:if` позволяет реализовывать условные конструкции. В отличие от инструкции `if` в большинстве языков программирования, инструкция `xsl:if` не содержит оператора `else`. Если проверка условия приведёт к логическому значению `true`, инструкция выполняется.

Примеры преобразований

- **Пример 1. Использование инструкции xsl:for-each.**

Исходный XML

```
<source>
<AAAid="a1" pos="start">
     <BBBid="b1"/>
     <BBBid="b2"/>
</AAA>
<AAAid="a2">
     <BBBid="b3"/>
     <BBBid="b4"/>
     <CCCid="c1">
          <DDDid="d1"/>
     </CCC>
     <BBBid="b5">
          <CCCid="c2"/>
     </BBB>
</AAA>
</source>
```

Преобразование XSLT

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

<xsl:template match="/">
     <xsl:for-each select="//BBB">
          <DIVstyle="color:red">
               <xsl:value-of select="name()"/>
               <xsl:text> id=</xsl:text>
               <xsl:value-of select="@id"/>
          </DIV>
     </xsl:for-each>
     <xsl:for-each select="source/AAA/CCC">
          <DIVstyle="color:navy">
               <xsl:value-of select="name()"/>
               <xsl:text> id=</xsl:text>
               <xsl:value-of select="@id"/>
          </DIV>
     </xsl:for-each>
</xsl:template>


</xsl:stylesheet>
```

Результат

```
<DIV style="color:red">BBB id=b1</DIV>
<DIV style="color:red">BBB id=b2</DIV>
```

```
<DIV style="color:red">BBB id=b3</DIV>
<DIV style="color:red">BBB id=b4</DIV>
<DIV style="color:red">BBB id=b5</DIV>
<DIV style="color:navy">CCC id=c1</DIV>
```

Представление HTML

BBB id=b1
BBB id=b2
BBB id=b3
BBB id=b4
BBB id=b5
CCC id=c1

- **Пример 2. Использование сортировки.**

Исходный XML

```
<source>
<name>John</name>
<name>Josua</name>
<name>Charles</name>
<name>Alice</name>
<name>Martha</name>
<name>George</name>
</source>
```

Преобразование XSLT

```
<xsl:stylesheet version = '1.0'
     xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="/">
     <TABLE>
          <xsl:for-each select="//name">
               <xsl:sort order="ascending" select="."/>
               <TR>
                    <TH>
                         <xsl:value-of select="."/>
                    </TH>
               </TR>
          </xsl:for-each>
     </TABLE>
</xsl:template>
</xsl:stylesheet>
```

Результат

```
<TABLE><TR>
     <TH>Alice</TH>
  </TR>
```

```
  <TR>
     <TH>Charles</TH>
  </TR>
  <TR>
     <TH>George</TH>
  </TR>
  <TR>
     <TH>John</TH>
  </TR>
  <TR>
     <TH>Josua</TH>
  </TR>
  <TR>
     <TH>Martha</TH>
  </TR>
</TABLE>
```

Представление HTML

Alice
Charles
George
John
Josua
Martha

- **Пример 3. Создание атрибута.**

Исходный XML

```
<source>
<color>blue</color>
<color>green</color>
<color>red</color>
</source>
```

Преобразование XSLT

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

<xsl:template match="color">
     <TABLE>
          <TR>
               <TD>
                    <xsl:attribute name="style">
                         <xsl:text>color:</xsl:text>
                         <xsl:value-of select="."/>
                    </xsl:attribute>
                    <xsl:value-of select="."/>
               </TD>
```

```
          </TR>
     </TABLE>
</xsl:template>
```

Результат

```
<TABLE>
  <TR>
     <TD style="color:blue">blue</TD>
  </TR>
</TABLE>
<TABLE>
  <TR>
     <TD style="color:green">green</TD>
  </TR>
</TABLE>
<TABLE>
  <TR>
     <TD style="color:red">red</TD>
  </TR>
</TABLE>
```

Представление HTML

blue
green
red

- **Пример 4. Использование инструкции xsl:if, строковых функций и переменных.**

Исходный XML

```
<source>
<car>белый BMW</car>
<car>BMW соспецсигналом</car>
<car>новая AUDI</car>
</source>
```

Преобразование 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="source">
<html><body>
      <xsl:variable name="total" select="count(car)"/>
      <b>СписокBMW:</b>
      <xsl:for-each select="car">
            <xsl:if test = "contains(.,'BMW')">
            <p><xsl:value-of select = "."/></p>
```

```
            </xsl:if>
      </xsl:for-each>
      <b>Всегомашин: </b><xsl:value-of select = "$total"/>
</body></html>
</xsl:template>
</xsl:stylesheet>
```

Результат

```
<html><body>
      <b>СписокBMW:</b>
      <p>белый BMW</p>
      <p>BMW со спецсигналом</p>
      <b>Всего машин: </b>3
</body></html>
```

Представление HTML

Список BMW:
белый BMW
BMW со спецсигналом
Всего машин: 3

Задание №1 по работе

1) Выполните задание 2а предыдущей лабораторной работы, на этот раз используя инструкцию *xsl:for-each* и применив сортировку по году выпуска для DVD и по названию для Blue-ray.
   Полученный после преобразования документ **html** представлен на рисунке 5.1

DVD

Название	Режиссёр	Страна	Жанр	Год
Операция «Ы» и другие приключения Шурика	Леонид Гайдай	СССР	комедия	1965
Иван Васильевич меняет профессию	Леонид Гайдай	СССР	комедия	1973
Форрест Гамп	Роберт Земекис	США	драма	1994
Зеленая миля	Фрэнк Дарабонт	США	фантастика	1999
Гладиатор	Ридли Скотт	США, Великобритания	боевик	2000
Престиж	Кристофер Нолан	США, Великобритания	триллер	2006

Blue-ray

Название	Режиссёр	Страна	Жанр	Год
Как приручить дракона	Дин ДеБлуа, Крис Сандерс	США	мультфильм	2010
Леон	Люк Бессон	Франция	триллер	1994
Начало	Кристофер Нолан	США, Великобритания	фантастика	2010
Шерлок Холмс	Гай Ричи	США, Германия	триллер	2009

Рисунок 5.1 Отображение **html** документа после преобразования.

2) Дополните исходный документ XML информацией о продолжительности каждого фильма.
3) Сведите информацию о продолжительности фильмов в следующую таблицу:

Тип носителя/Количество/Общая продолжительность/Средняя продолжительность

При этом в столбце «Тип носителя» вывести информацию для DVD, Blue-ray, а также суммарную информацию.

Подсказка: используйте функцию sum() для суммирования продолжительности фильмов. Символ "/" зарезервирован для путей адресации, используйте ключевое слово *div* для операции деления.

Полученный документ **html** представлен на рисунке 5.2

Информация о продолжительности фильмов

Тип носителя	Количество	Общая продолжительность	Средняя продолжительность
DVD	6	810	135
Blue-ray	4	484	121
Все носители	10	1294	129.4

Рисунок 5.2 Отображение **html** документа после преобразования.

4) Выведите названия фильмов, удовлетворяющие поочерёдно следующим условиям:

- в создании фильма участвовали США (или страна на Ваш выбор)

- в создании фильма участвовали несколько стран
- продолжительность фильма менее двух часов
- фильм выпущен после 2000 года

Полученный документ **html** представлен на рисунке5.3.

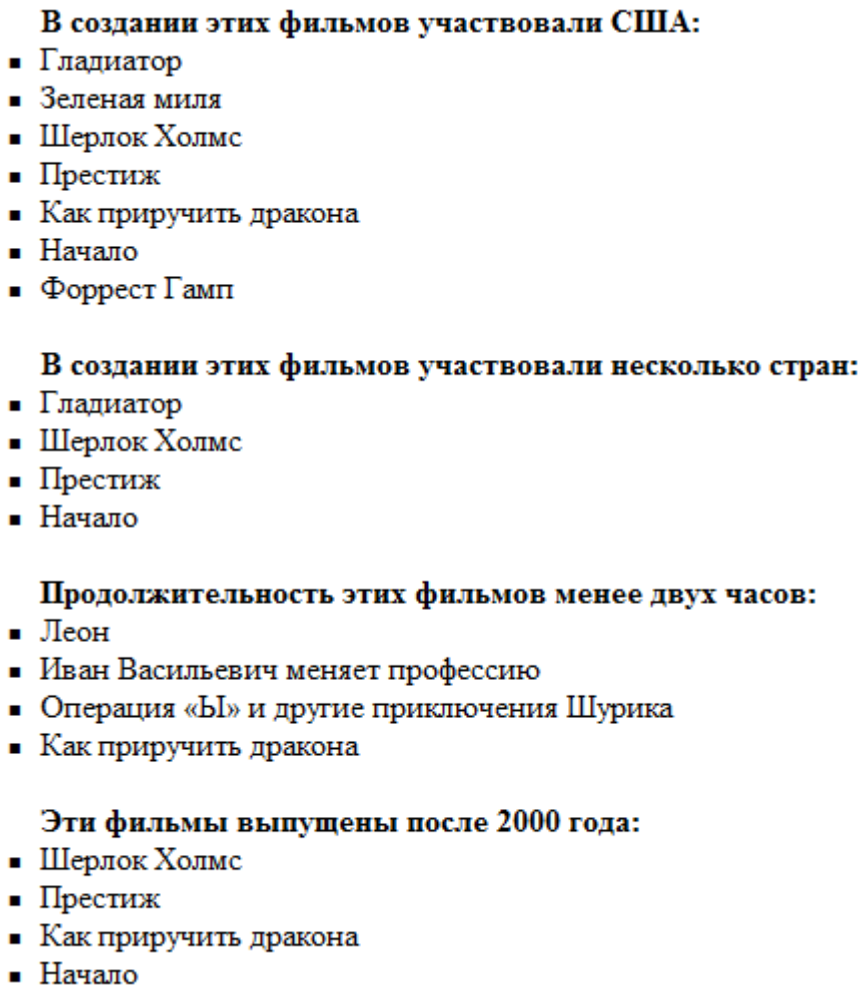


Рисунок 5.3. Отображение **html** документа после преобразования.

Задание №2 по работе

Создайте такой шаблон XSL, который бы менял структуру исходного документа **xml**. Пускай теперь вложенные в корневой элемент теги должны иметь одинаковые названия, при этом тип носителя будет храниться в атрибуте:
<фильм носитель = “DVD”>

Страна-создатель теперь нам не важна. Год выпуска должен быть заключён в содержимом тега, а не быть атрибутом:
<год>2000</год>

А тег «продолжительность» напротив теперь является пустым с единственным атрибутом:
<продолжительность мин = “171” />

Вид нового документа **xml** в браузере показан на рисунке5.4, некоторые элементы свёрнуты для экономии места.

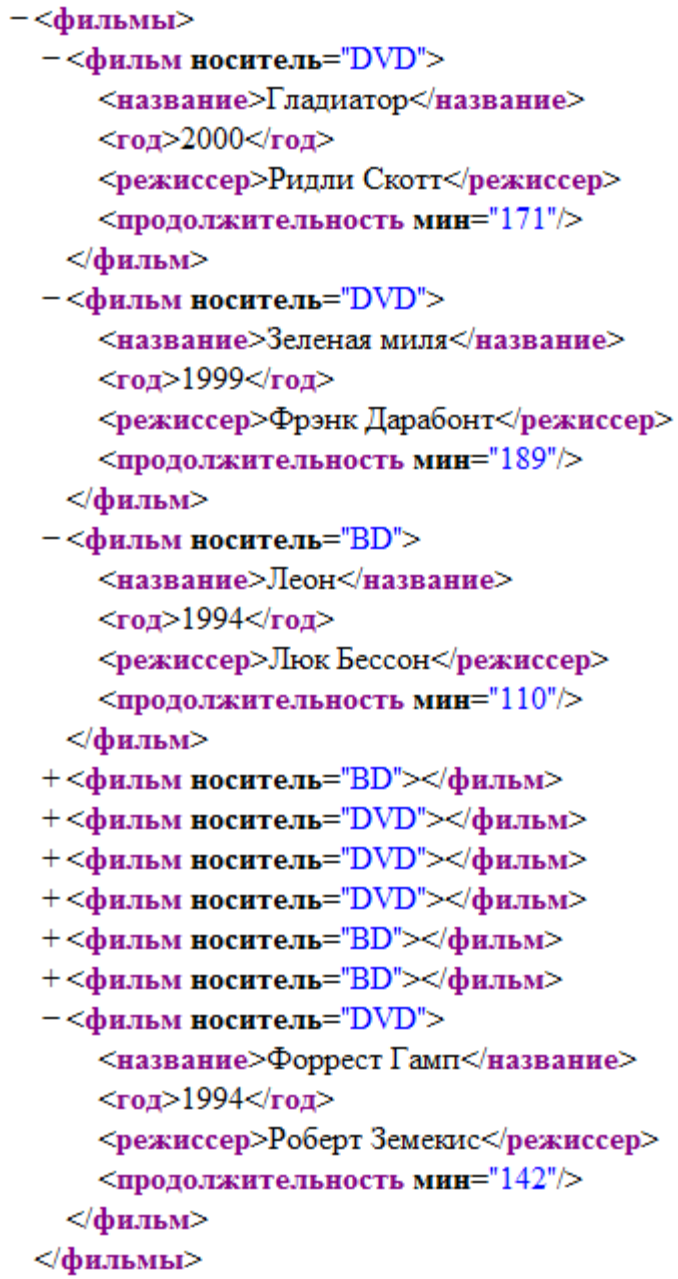


Рисунок 5.4. Отображение нового **xml** документа в браузере.

4 Содержание отчёта

- Описание цели лабораторной работыиизученных возможностей XSLT
- Предоставление правильно построенных шаблонов, выполняющих необходимые преобразования

Лабораторная работа № 6
«Объектная модель документа XML - DOM»

Цели лабораторной работы

Целью работы является получение сведений об объектной модели XML – DOM, способах реализации DOM, а также выполнение простейших сценариев для документа XML с помощью javascript.

Содержание работы

- DOM для XML
- Свойства, методы и события объектов DOM
- Примеры сценариев DOM
- Задания для самостоятельной работы

Порядок выполнения работы

DOM для XML

Объектная модель документа DOMдляXML предоставляет общий прикладной интерфейс API, спецификацию консорциума W3C. XMLDOM даёт стандартизированный способ доступа к информации, содержащийся в документах XML, и манипулированию ею, описывая стандартные свойства, события и методы для взаимодействия с документами XML.

Модель DOM не накладывает ограничений на структуру документа. Любой документ известной структуры с помощью DOM может быть представлен в виде дерева узлов, каждый узел которого представляет собой элемент, атрибут, текстовый, графический или любой другой объект. Узлы связаны между собой отношениями родительский-дочерний.

DOM обеспечивает и использует организационную структуру объектов и сценариев для доступа к её узлам, а также манипулирования ими. Сценарий может указывать на узел по его относительному или абсолютному положению, например первый узел в структуре документа. Сценарии также способны добавлять или удалять узлы. Благодаря этому содержание каждого узла можно получить или обновить как часть приложения. Таким образом, объекты DOM представляют собой хорошо сконструированные, уникальным образом идентифицированные контейнеры. Их поведение можно расширить с помощью сценариев.

Объект *Document* – это корневой узел-контейнер для всех остальных объектов, определённых в документе XML. Этот узел предоставляет доступ к определениям типа документа DTD, а также определяет дочерние элементы в структуре документа. Поскольку все объекты в дереве документа можно идентифицировать, Вы можете ссылаться как на них, так и на их содержание. Например, JavaSctipt позволяет Вам сослаться на текстовое содержание первого дочернего узла в структуре документа следующим образом:

mystuff.documentType.childNodes.item(0).text

DOMможет реализовываться различными способами с помощью синтаксических анализаторов, поддерживающих обработку DOM. При выполнении упражнений этой работы обработчик MSXML используется совместно с JavaScript с помощью InternetExplorer. Однако не все посетители Web-узла могут иметь необходимое программное обеспечение. Для решения подобных проблем созданы реализации DOM, поддерживающие различные обработчики и сценарии. Существую реализации DOM, выполненные на PERL, Java, Pyton и нескольких других популярных языках программирования, поддерживающих большое количество платформ.

Свойства, методы и события объектов DOM

Модель XML DOM компании Microsoft содержит четыре основных объекта: *XMLDOMDocument, XMLDOMNode, XMLDOMNodeList, XMLDOMNameNodeMap*. Другие реализации XML DOM используют другие объекты, зависящие от языков реализации. Каждый из объектов обладает своими свойствами, методами, событиями.

Объект *XMLDOMDocument* будет активно использовать в этой лабораторной работе. Некоторые из часто используемых свойств, методов и событий перечислены ниже.

Свойства XMLDOMDocument

- async (возможность асинхронной загрузки, принимает значения true или false)
- attributes (список атрибутов)
- childNodes(список дочерних узлов)
- docType (DTD схема)
- documentElement (корневой узел)
- firstChild(1-ой дочерний элемент)
- lastChild(последний дочерний элемент)
- namespaceURI(пространство имен)
- nodeName (имя элемента, если имеется)
- nodeType (тип узла)
- nodeValue (текст, связанный с узлом)
- parseError (сведение о последней ошибке)
- preserveWhiteSpace (требует свободного пространство)
- readyState (текущее состояние документа XML)
- resolveExternals (решает пространства имен, схемы DTD и внешние ссылки во время обработки документа)
- validateOnParse (проверка документа при обработке)
- xml (содержит представление XML узла и всех его потомков)

Методы XMLDOMDocument

- Clonenode() – создает копию текущего узла
- CreateAttribute () – создает новый атрибут
- CreateCDATASec() – создает узел раздела CDATA
- CreateElement() – создаетузел элемента
- CreateComment() – создаетузел комментария
- CreateEntityReference() – создаетобъект ссылки
- CreateNode() – создает узел
- CreateTextNode()– создаеттекстовой узел
- GetElementsByTagName() – возвращаетэлементы с указанным именем
- NodeFromId() – возвращаетузел с указанным ID
- HasChildNodes() –trueесли содержит дочерние узлы
- Load() – загружаетдокумент XML
- LoadXML() – загружаетдокумент XML из указанной строки
- RemoveChild() – удаляетуказанный дочерний элемент
- ReplaceChild() – заменяетуказанный дочерний элемент
- Save() – сохраняет документ XML

События XMLDOMDocument

- ondataavailiable - данные документа доступны
- onreadystatechange - свойство readyState изменилось

Примеры сценариев DOM

- **Пример 1. Загрузка документа.**

Фрагментjavascript

```
<head>
<script language=”javascript”>
varoMystuff = new ActiveXObject(“Microsoft.XMLDOM”)
oMystuff.async=“false”
oMystuff.load(“somedocument.xml”)
</script>
</head>
```

- **Пример 2. Отображение текстового содержания документа.**

Исходный XML

```
<?xml version="1.0" encoding="windows-1251"?>
<note>
        <message ID="m1" from="mom">
                Remember to buy milk on the way home from work
        </mesage>
        <message ID="m2" from="sister">
                I need some help with my homework
        </mesage>
        <message ID="m3" from="gf">
                Please play Scrabble with me tonight
        </mesage>
</note>
```

Фрагментjavascript…

```
</head>
<script language=”javascript”>
varoMystuff = new ActiveXObject(“Microsoft.XMLDOM”)
oMystuff.async=“false”
oMystuff.load(“msg.xml”)
alert(oMystuff.text);
</script>
```

Результат работы скрипта представлен на рисунке 6.1.

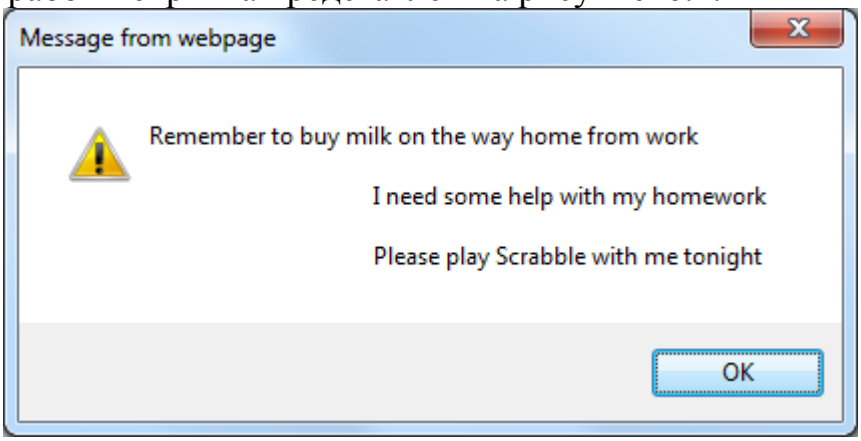


Рисунок 6.1. Сообщение браузера.

- **Пример 3. Выбор узла по его расположению.**

Фрагментjavascript

```
…
</head>
<script language="javascript">
document.write("<h2>The selected XML element if the file contains:</h2>")
document.write(oMystuff.documentElement.childNodes.item(1).text)
document.write("<hr/>")
</script>
```

Результат работы скрипта представлен на рисунке 6.2.

Рисунок 6.2. Вывод второго элемента на экран.

- **Пример 4. Выбор элемента по имени.**

Фрагментjavascript

```
…
</head>
<script language="javascript">
document.write("<h2>Result by Tag Name:</h2>")
document.write(oMystuff.getElementsByTagName("message").item(1).text)
document.write("<hr/>")
</script>
```

Результат работы скрипта представлен на рисунке 6.3.

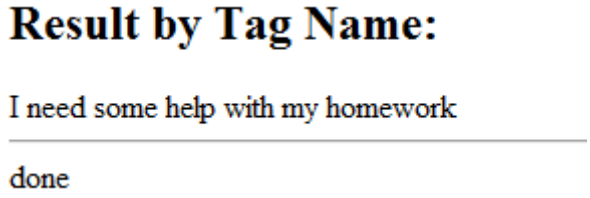


Рисунок 6.3. Вывод элемента по имени на экран.

Задания по работе

Напишите сценарий, который будет инициализировать объектную модель DOM. Загрузите документ XML, разработанный в предыдущих работах, в объект документ и отобразите в окне браузера содержание элементов <название> и <режиссер> для первого, второго и четвёртого фильмов коллекции.

Измените код таким образом, чтобы в окне браузера отображалась информация о первом фильме на носителе Blue-ray из коллекции. Выведите информацию о названии фильма, режиссёре и стране.

Содержание отчёта

- Описание цели лабораторной работы и полученных сведений об объектной модели XML – DOM
- Предоставление работающихсценариев

Литература

1. Деван Шепард, Освой самостоятельно XML за 21 день, 2-е издание. Пер. с англ. – М.: Издательский дом «Вильямс», 2002. – 432 с.
2. Спецификация Extensible Markup Language (XML) 1.0 (Second Edition) (W3C Recommendation). Пер. с англ. – Усманов Р., Luxoft // http://www.rol.ru/news/it/helpdesk/xml01.htm
3. Спецификация XSL Transformations (XSLT) Version 1.0 (W3C Recommendation). Пер. с англ. – Усманов Р., Luxoft // http://www.rol.ru/news/it/helpdesk/xslt01.htm
4. Печерский А., Язык XML – практическое введение – электронная статья // http://www.codenet.ru/webmast/xml/
5. MiloslavNic, XSLT в примерах – электронный учебник. Пер. с англ. Ярошевич В. // http://www.zvon.org/xxl/XSLTutorial/Output_rus/index.htm