

# Лекция 11

# Листы стилей XML

Язык XSL позволяет преобразовывать XML-документы в документы других типов, а также задает порядок их форматирования.

Принципы преобразования XML-документов описаны в разделе XSL Transformations (XSLT) спецификации XSL.

Средства, определяемые XSLT, позволяют конвертировать XML-код в HTML, PDF, ASCII-текст и другие типы документов.

Для форматирования документов в языке XSL предусмотрены специальные объекты и свойства. Они сообщают приложению о том, как следует разместить элементы XML-документа на странице.

Элементы, применяемые для форматирования, имеют стандартные имена, перед которыми указывается префикс пространства имен **fo**:

Приложение, поддерживающее XML и XSL, читает документ и связанный с ним лист стилей, а затем преобразует документ по правилам заданным с помощью XSL-выражений.

XSL-приложение представляет содержимое XML-документа в виде древовидной структуры, а затем по этой структуре строит целевой документ.

В частности, целевым может быть неструктурированный документ например HTML-страница или текстовый файл.

Листы стилей XSL, как и документ, с которым они связываются, представляются формате XML.

Листы стилей содержат шаблоны, описывающие древовидную структуру исходного документа.

Корневой элемент листа стилей XSL с именем **xsl:stylesheet** содержит атрибут **xmlns:xsl**

В качестве шаблонов выступают элементы именем **xsl:template**.

Элементы **xsl:template** с атрибутами определяют правила, по которым устанавливается соответствие между элементами XML и шаблонами листа стилей.

Инструкции по обработке фрагментов, соответствующих шаблонам, задаются с помощью подчиненных элементов, входящих в состав **xsl:template**

Имена этих подчиненных элементов начинаются с префикса `xsl:.`

Остальные компоненты элементов `xsl:template` представляют XML-выражения и данные, используемые при формировании выходного XML-документа.

Формат листа стилей XSL имеет вид:

**<?xml version="1.0"?>**

**<xsl:stylesheet**

**xmlns:xsl="http://www.w3.org/TR/WD-xsl">**

**<xsl:template *match*="имя\_элемента">**

*<!-- Действия для элемента с именем  
имя\_элемента -->*

*<!-- выбор подчиненного элемента с именем  
подчиненный\_элемент -->*

**<xsl:apply-template**

**select="подчиненный\_элемент"/>**

*<!-- Указание стиля -->*

**</xsl:template>**

**</xsl:stylesheet>**

Ссылка на лист стилей XSL, содержащаяся в XML-документе, имеет следующий вид.

```
<?xml version="1.0"?>
```

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

.....

Рассмотрим пример с использованием XML.

Создадим XML файл my.xml

```
<?xml version="1.0"?>
```

```
<steps>
```

```
<step>
```

```
<name>Step1</name>
```

```
</step>
```

```
<step>
```

```
<name>Step2</name>
```

```
</step>
```

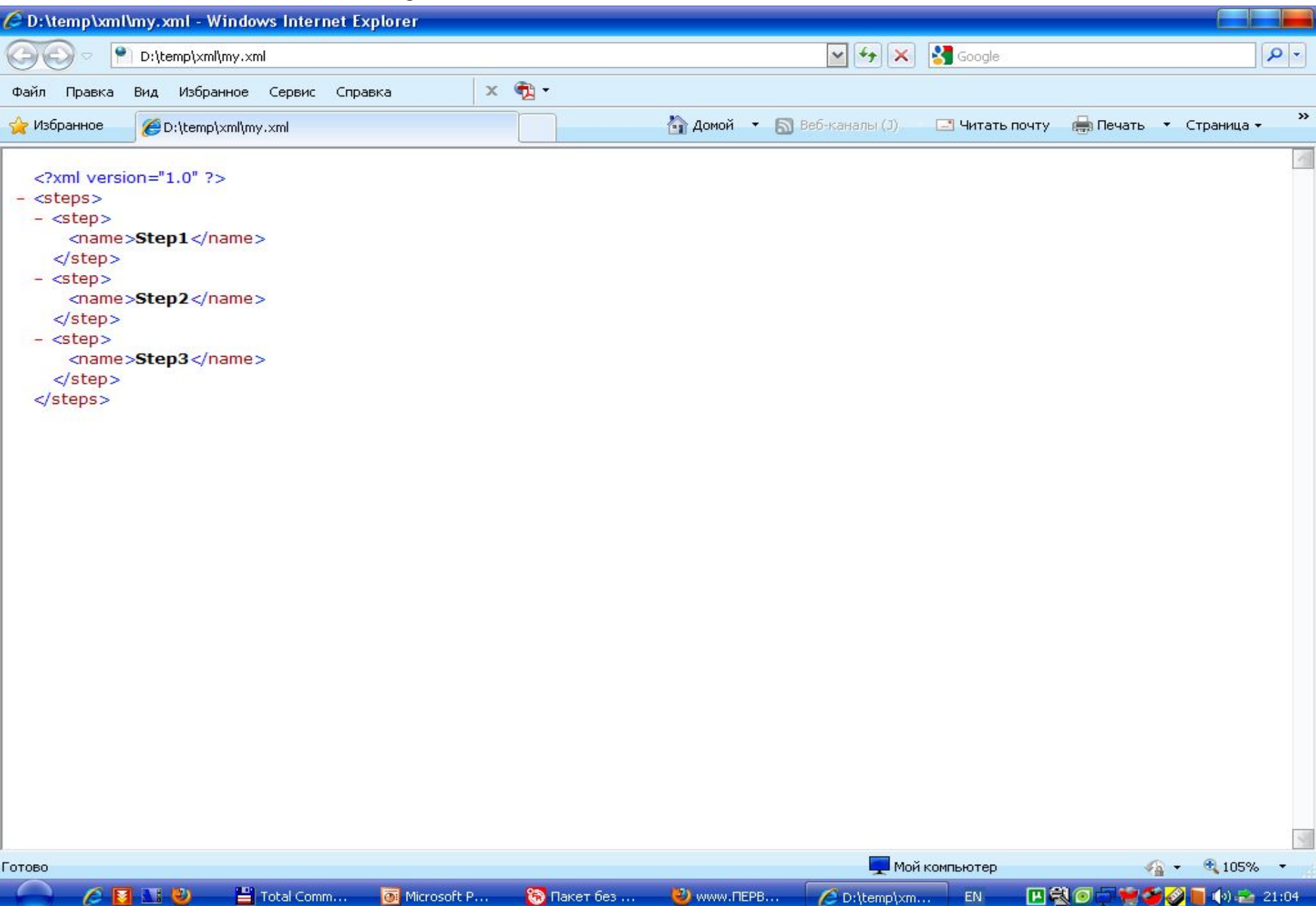
```
<step>
```

```
<name>Step3</name>
```

```
</step>
```

```
</steps>
```

# На экране браузера имеем



The screenshot shows a Windows Internet Explorer browser window. The title bar reads "D:\temp\xml\my.xml - Windows Internet Explorer". The address bar contains "D:\temp\xml\my.xml". The menu bar includes "Файл", "Правка", "Вид", "Избранное", "Сервис", and "Справка". The toolbar shows "Избранное", "Домой", "Веб-каналы (J)", "Читать почту", "Печать", and "Страница". The main content area displays the following XML code:

```
<?xml version="1.0" ?>
- <steps>
- <step>
  <name>Step1</name>
</step>
- <step>
  <name>Step2</name>
</step>
- <step>
  <name>Step3</name>
</step>
</steps>
```

The taskbar at the bottom shows the system tray with "Готово", "Мой компьютер", and "105%". The taskbar includes icons for "Total Comm...", "Microsoft P...", "Пакет без ...", "www.ПЕРВ...", "D:\temp\xm...", and "EN". The system clock shows "21:04".



При отображении xml файла можно использовать технологию CSS. Тогда в файл

my.xml будет иметь вид

```
<?xml version="1.0"?>
```

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

```
<steps>
```

```
<step>
```

```
<name>Step1</name>
```

```
</step>
```

```
<step>
```

```
<name>Step2</name>
```

```
</step>
```

```
<step>
```

```
<name>Step3</name>
```

```
</step>
```

```
</steps>
```

Файл my.css будет иметь вид

step

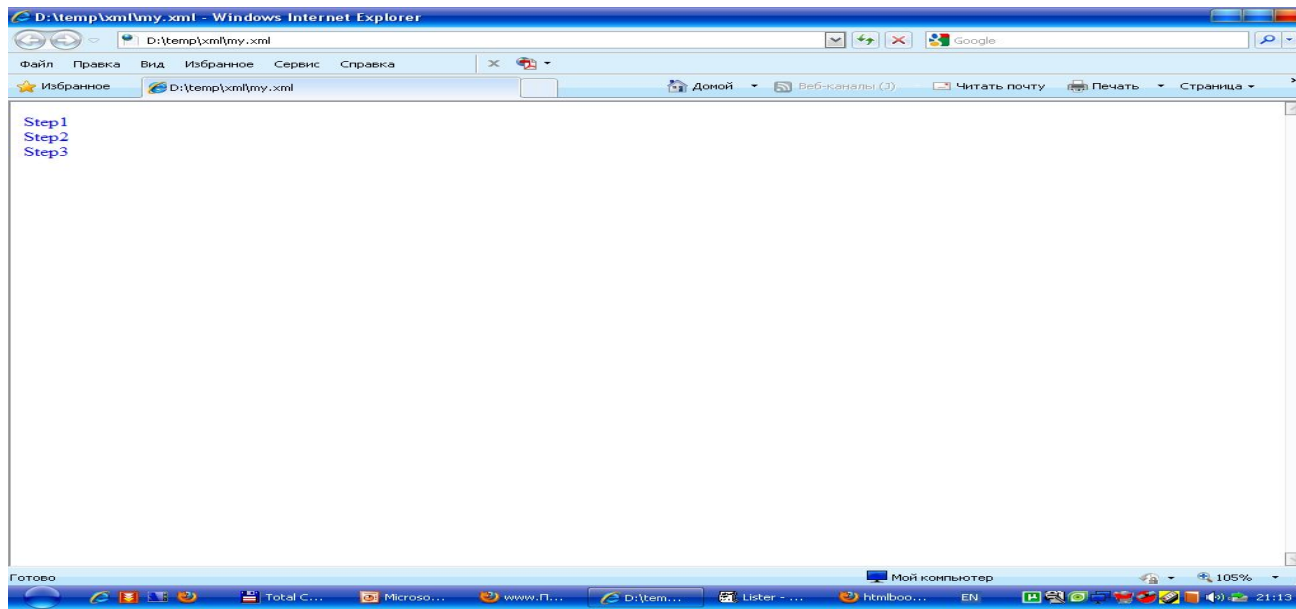
{

COLOR: blue;

DISPLAY: block

}

На экране получим



XML файл с DTD определением имеет вид

```
<?xml version="1.0"?>
  <!DOCTYPE step
  [
    <!ENTITY home "www.firststeps.ru">
  ]
  >
<steps>
  <step>
    <name>Step1</name>
    <author>&home;</author>
  </step>
<step>
  <name>Step2</name>
  <author>&home;</author>
</step>
<step>
  <name>Step3</name>
  <author>&home;</author>
</step>
</steps>
```

# В браузере этот файл имеет вид

D:\temp\xml\my.xml - Windows Internet Explorer

D:\temp\xml\my.xml

Файл Правка Вид Избранное Сервис Справка

Избранное D:\temp\xml\my.xml Домой Веб-каналы (J) Читать почту Печать Страница

```
<?xml version="1.0" ?>
<!DOCTYPE step (View Source for full doctype...)>
- <steps>
- <step>
  <name>Step1</name>
  <author>www.firststeps.ru</author>
</step>
- <step>
  <name>Step2</name>
  <author>www.firststeps.ru</author>
</step>
- <step>
  <name>Step3</name>
  <author>www.firststeps.ru</author>
</step>
</steps>
```

Готово Мой компьютер 105% 21:24

Рассмотрим использование вместо CSS XSL.

Для этого в XML файле сделаем ссылку на файл my.xsl

```
<?xml version="1.0"?>  
<?xml-stylesheet type="text/xsl" href="my.xsl"?>  
  <note>  
    <head>Step1</head>  
    <head>Step2</head>  
  </note>
```

Файл my.xsl имеет вид

```
<?xml version="1.0"?>  
  <xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">  
    <xsl:template match="/">  
      <html>  
        <body>  
          <H1>Hello</H1>  
          <span style="color:red"> <xsl:apply-templates/> </span>  
        </body>  
      </html>  
    </xsl:template>
```

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

```
  <h1>Hello2</h1>
```

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

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

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

```
  <h1>Hello1</h1>
```

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

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

```
<xsl:template match="text()">
```

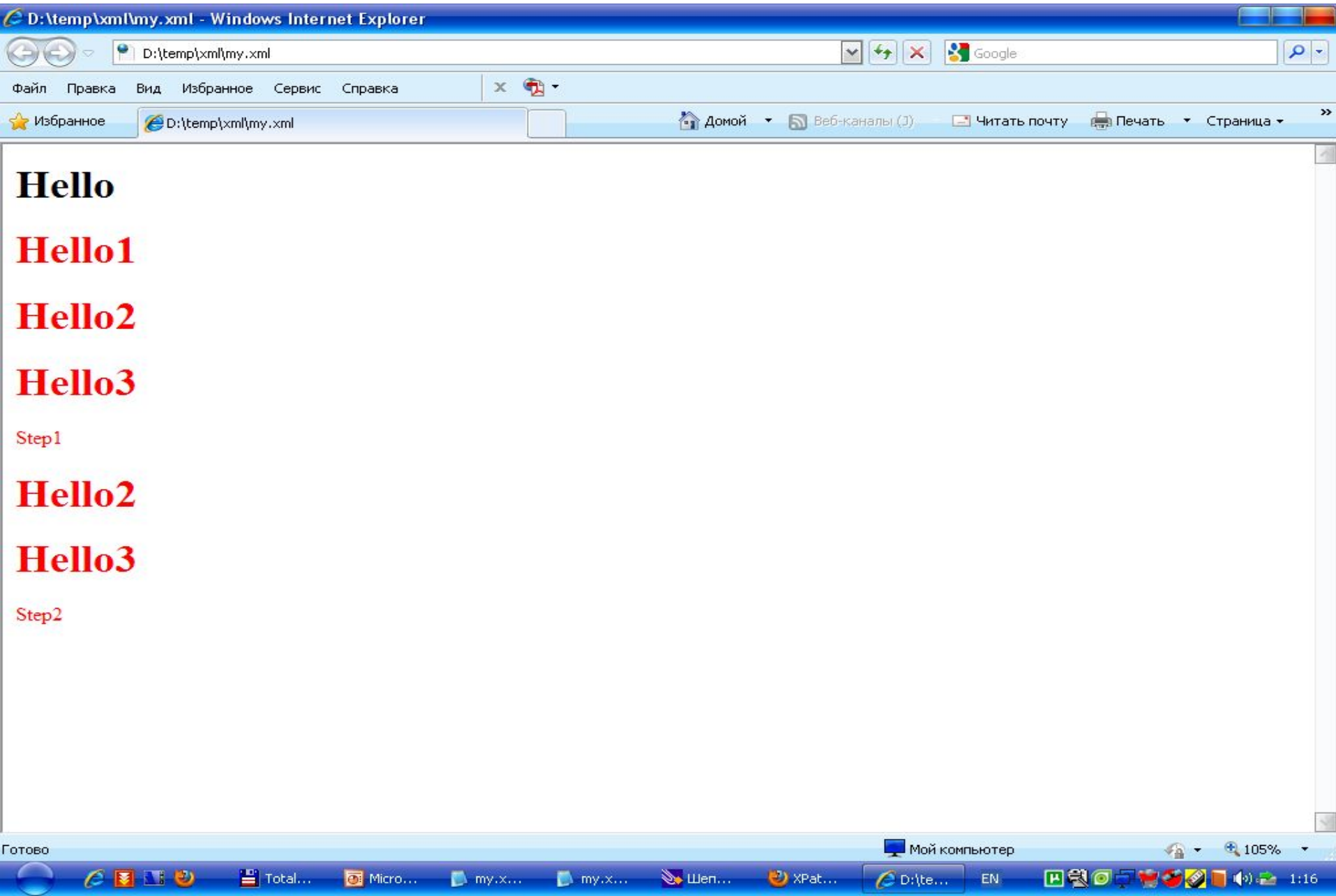
```
  <h1>Hello3</h1>
```

```
  <xsl:value-of select="."/> Вставка значения выбранного  
узла в виде текста
```

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

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

# Браузер отобразит следующее



# XML-анализаторы

Каждое приложение, работающее с XML, использует анализатор, который представляет собой некоторый компонент, находящийся между приложением и файлами XML.

Документы XML могут быть либо well-formed, либо valid. Документы wellformed составлены в соответствии с синтаксическими правилами построения XML-документов. Документы не только сформированы синтаксически правильно, но и следуют некоторой структуре, которая описана в DTD.

Соответственно есть валидирующие и невалидирующие анализаторы. И те, и другие проверяют XML-документ на соответствие синтаксическим правилам, но только валидирующие анализаторы знают, как проверить XML-документ на соответствие структуре, описанной в DTD.



Никакой связи между видом анализатора и видом XML-документа нет.

Валидирующий анализатор может разобрать XML-документ, для которого нет DTD, и, наоборот, невалидирующий анализатор может разобрать XML-документ, для которого есть DTD.

Существует два вида взаимодействия приложения и анализатора: использовать модель, основанную на представлении содержимого файла XML в виде дерева объектов, либо событийную модель.

Анализаторы, которые строят древовидную модель, – это DOM-анализаторы (Dynamic Object Model).  
Анализаторы, которые генерируют события, – это SAX-анализаторы (Simple API for XML).

**В первом случае** анализатор строит в памяти дерево объектов, соответствующее XML-документу. Далее вся работа ведется именно с этим деревом.

**Во втором случае** анализатор работает следующим образом: когда происходит анализ документа, анализатор генерирует события, связанные с различными участками XML-файла, а программа, использующая анализатор, решает, как реагировать на эти события.

Так, анализатор будет генерировать событие о том, что он встретил начало документа либо его конец, начало элемента либо его конец, символьную информацию внутри элемента и т.д.

**DOM-анализаторы** следует использовать тогда, когда нужно знать структуру документа и может понадобиться изменять эту структуру либо использовать информацию из XML-файла несколько раз.

**SAX-анализаторы** используются тогда, когда нужно извлечь информацию о нескольких элементах из XML-файла либо когда информация из документа нужна только один раз.

# SAX-анализаторы

SAX API определяет ряд событий, которые будут сгенерированы при разборе документов:

**startDocument** – событие, сигнализирующее о начале документа;

**endDocument** – событие, сигнализирующее о завершении документа;

**startElement** – данное событие будет сгенерировано, когда анализатор полностью обработает содержимое открывающего тега, включая его имя и все содержащиеся атрибуты;

**endElement** – событие, сигнализирующее о завершении элемента;

**characters** – событие, сигнализирующее о том, что анализатор встретил символьную информацию внутри элемента;

**warning, error, fatalError** – эти события сигнализируют об ошибках при разборе XML-документа.

В пакете **org.xml.sax.helpers** содержится класс **DefaultHandler**, который содержит методы для обработки всех вышеуказанных событий.

Для создания приложения обрабатывающего XML файл необходимо:

1. Создать класс, суперклассом которого будет **DefaultHandler**, и переопределить методы, отвечающие за обработку интересующих событий.
2. Создать объект-парсер класса **org.xml.parsers.SAXParser**.
3. Вызвать метод **parse()**, которому в качестве параметров передать имя разбираемого файла и экземпляр созданного на первом шаге класса.

Рассмотрим пример разбор документа notepad.xml, который имеет вид

```
<?xml version="1.0"?>  
  <!DOCTYPE notepad SYSTEM "notepad.dtd">  
<notepad>  
  <note login="rom">  
    <name>Valera</name>  
    <tel>217819</tel>  
    <url>http://www.b.com</url>  
    <address>  
      <street>Main Str., 35</street>  
      <city>Kiev</city>  
      <country>UKR</country>  
    </address>  
  </note>
```

```
<note login="goch">
```

```
<name>Igor</name>
```

```
<tel>430797</tel>
```

```
<url>http://www.a.com</url>
```

```
<address>
```

```
<street>Deep Forest, 7</street>
```

```
<city>Polock</city>
```

```
<country>VCL</country>
```

```
</address>
```

```
</note>
```

```
</notepad>
```



```
import org.xml.sax.Attributes;  
import org.xml.sax.helpers.DefaultHandler;  
import javax.xml.parsers.SAXParser;  
import javax.xml.parsers.SAXParserFactory;  
import java.net.URL;  
import java.net.MalformedURLException;  
import java.util.Vector;
```

```
interface ConstNote
```

```
{  
    int NAME = 1, TEL = 2, URL = 3,  
    STREET = 4, CITY = 5, COUNTRY = 6;  
}
```

```
class DocHandler extends DefaultHandler
                                implements ConstNote{
    Vector notes = new Vector();
    Note curr = new Note();
    int current = -1;
public Vector getNotes() { return notes; }
public void startDocument() {
    System.out.println("parsing started"); }
public void endDocument(){System.out.print("");}
```

```
public void startElement(String uri, String localName,
    String qName, Attributes attrs) {
    if (qName.equals("note")) {
        curr = new Note();
        curr.setLogin(attrs.getValue(0));}
    if (qName.equals("name")) current = NAME;
    else if (qName.equals("tel")) current = TEL;
        else if (qName.equals("url")) current = URL;
            else if (qName.equals("street"))
                current = STREET;
                else if (qName.equals("city"))
                    current = CITY;
                    else if (qName.equals("country"))
                        current = COUNTRY;
    }
}
```

```
public void endElement(String uri, String localName,
    String qName){
    if (qName.equals("note")) notes.add(curr);}
public void characters(char[] ch, int start, int length) {
String s = new String(ch, start, length);
    try{
        switch (current) {
            case NAME: curr.setName(s); break;
            case TEL: curr.setTel(Integer.parseInt(s)); break;
            case URL: try { curr.setUrl(new URL(s));}
                catch (MalformedURLException e) {}; break;
            case STREET:curr.address.setStreet(s); break;
            case CITY: curr.address.setCity(s); break;
            case COUNTRY: curr.address.setCountry(s);break;}
        }
    catch (Exception e) { System.out.println(e);} } }
```

```
public class MyParserDemo {  
    public static void main(String[] args) {  
try {  
    SAXParser parser =  
        SAXParserFactory.newInstance().newSAXParser();  
    DocHandler dh = new DocHandler();  
    Vector v;  
    if (dh != null) parser.parse("notepad.xml", dh);  
    v = dh.getNotes();  
    for (int i = 0; i < v.size(); i++)  
        System.out.println(((Note) v.elementAt(i)).toString());  
    } catch (Exception e) { e.printStackTrace();} } }
```

В результате на консоль будет выведена следующая информация:

**parsing started**

**rom**

**Valera 217819 <http://www.b.com>**

**address:Main Str., 35 Kiev UKR**

**goch**

**Igor 430797 <http://www.a.com>**

**address:Deep Forest, 7 Polock VCL**

Класс Note имеет вид:

```
import java.net.URL;
class Note {
    private String name, login;
    private int tel;
    private URL url;
    public Address address = new Address();
public void setAddress(Address address)
    { this.address = address; }
public void setLogin(String login)
    { this.login = login;}
public void setName(String name)
    { this.name = name; }
public void setTel(int tel)
    { this.tel = tel; }
```

```
public String toString() {  
    return login + " " + name + " " + tel + " "  
    + url + "\n\t address:" + address.street + " "  
    + address.city + " " + address.country; }  
class Address {  
    String street, city, country;  
public void setCity(String city) { this.city = city;}  
public void setCountry(String state) {  
    this.country = state; }  
public void setStreet(String street){  
    this.street = street;}  
}  
public void setUrl(URL url) { this.url = url; } }
```