Лабораторная работа № 6

DOM-анализатор

1. Основные понятия о DOM-классах

.NET-классы предоставляют два основных механизма для работы с XML-документами: классы `XmlReader` и `XmlWriter` – с одной стороны, и DOM-классы – с другой. В отличие от поточных классов `XmlReader` и `XmlWriter`, работающих в режиме «только вперед», DOM-модель предполагает единовременную загрузку в память сразу всего документа. Документ разбивается на отдельные составляющие: элементы, атрибуты, комментарии, инструкции обработки и т. п. Из этих составляющих строится древовидная структура документа (http://msdn.microsoft.com/ru-ru/library/hf9hbf87.aspx). Отдельные компоненты документа называются узлами, и каждый из них представляется соответствующей реализацией абстрактного класса `XmlNode` (http://msdn.microsoft.com/ru-ru/library/system.xml.xmlnode.aspx) из пространства имен System.Xml. Класс `XmlNode` реализует ядро основной модели объектов документов W3C (DOM) уровня 1 и уровня 2 (http://www.w3.org/standards/techs/dom#w3c_all). От класса `XmlNode` образован производный класс `XmlDocument` (http://msdn.microsoft.com/ru-ru/library/system.xml.xmldocument.aspx), который добавляет собственные методы и свойства, поддерживающие загрузку и сохранение документов, создание новых узлов и другие операции.

Существует несколько полезных эмпирических правил касающихся того, когда следует использовать классы `XmlReader` и `XmlWriter`, а когда - DOM-классы.

Классы `XmlReader` и `XmlWriter` лучше всего подходят:

a) Для одновременной обработки большого количества документов - это позволяет уменьшить объем занимаемой памяти
b) Для извлечения относительно небольших объемов информации из больших документов
c) Для работы с документами, обладающими простой структурой

Классы DOM лучше приспособлены:

a) Для одновременной обработки небольшого количества документов, а также для применения в тех ситуациях, когда объем используемой памяти и скорость обработки не являются критичными
b) Для извлечения всей или почти всей информации из больших документов
c) Для работы с документами, обладающими сложной структурой

В качестве примера рассмотрим следующий XML-документ:

```
<?xml version="1.0" encoding="utf-8"?>
<customer customerID="cust123">
  <customerName>Fred Q. Customer</customerName>
  <address>
    <streetAddress>314 Somewhere Street</streetAddress>
    <city>Randomsville</city>
    <state>XY</state>
    <postalCode>27182</postalCode>
  </address>
</customer>
```

Этот документ будет представлен DOM-классами в виде древовидной структуры, изображенной на диаграмме ниже.

Документ
    Элемент: `customer`
        Элемент: `customerName`
            Text: `Fred Q. Customer`

Элемент: address
Элемент: streetAddress
Text: 314 Somewhere Street
Элемент: city
Text: Randomsville
Элемент: state
Text: XY
Элемент: postalCode
Text: 27182
Элемент: customerID
Text: cust123

На этой диаграмме каждый уровень представляет узел или, если быть более точным, экземпляр класса, который реализует абстрактный класс `XmlNode`. Следует обратить внимание на то, что для содержимого, имеющего форму текста, существует свой собственный тип узлов. Поскольку содержимое элементов может быть смешанным, т. е. состоящим одновременно из текста и элементов, содержащийся текст должен представляться в виде отдельных узлов. Кроме того, следует обратить внимание на то, как атрибут `customerID` связан со всей остальной диаграммой. Этим подчеркивается тот факт, что узлы-атрибуты обрабатываются отличным образом от остальных узлов DOM-модели XML-документа.

2. Открытие уже существующего документа

Чтобы открыть уже существующий XML-документ, можно для созданного объекта `XmlDocument` вызвать метод `LoadXml()` или один из вариантов перегруженного метода `Load()`. Когда XML-документ анализируется объектом `XmlDocument`, то в соответствии с содержимым данного документа заполняется список `ChildNodes` (набор объектов `XmlNode`, которые описывают содержимое XML-документа); по мере необходимости этот список можно будет просматривать. Ниже приводятся наиболее распространенные способы открытия документов с использованием класса `XmlDocument`.

2.1. Открытие документа по URL

Чтобы открыть XML-документ по его URL, можно воспользоваться вариантом перегруженного метода `Load()`, принимающим в качестве параметра строку. Эта строка представляет собой URL, по которому данный XML-документ может быть найден. Например:

```
XmlDocument myDocument = new XmlDocument();
myDocument.Load("http://localhost/sample.xml");
// далее следует соответствующая обработка документа
```

2.2. Открытие документа, находящегося в файле

Для открытия документа, находящегося в файле, сначала необходимо создать объект `FileStream` на базе данного файла. Этот объект передается методу `Load()` объекта `XmlDocument` в качестве параметра. Затем документ загружается из этого потока и анализируется. Например:

```
XmlDocument myDocument = new XmlDocument();
FileStream myFile = new FileStream("mydoc.xml", FileMode.Open);
myDocument.Load(myFile);
// далее следует соответствующая обработка документа
```

2.3. Открытие документа, находящегося в виде строки в памяти

Чтобы выполнить анализ документа, который хранится в памяти в виде строки, можно воспользоваться методом `LoadXml()` объекта `XmlDocument`. Он интерпретирует строку как XML-документ и соответствующим образом анализирует ее. Этот метод может быть применен, например, для анализа XML-

строки, передаваемой в качестве параметра SOAP-сообщения, или для строки, извлекаемой из реляционной базы данных. Например:

```
XmlDocument myDocument = new XmlDocument();
String myXML = "<customer><name>Fred Q. Somebody</name></customer>";
myDocument.Load(myXML);
// далее следует соответствующая обработка документа
```

Просмотр XML-документа, загруженного в DOM-классы, выполняется так. Для каждого узла документа имеется список относящихся к нему дочерних узлов – семейство `ChildNodes`. Кроме того, существует ссылка на узел-родитель данного узла, хранящаяся в свойстве `ParentNode`. В зависимости от типа узла можно определить тип содержимого и извлечь его из документа.

3. Поиск информации, содержащейся в документе

Чтобы найти конкретный узел или группу узлов нет необходимости просматривать все узлы документа. Методы `GetElementsByTagName`, `GetElementsById`, `SelectNodes` и `SelectSingleNode` класса `XmlDocument` позволяют отобрать нужные узлы. В результате использования этих методов обычно получают объект `XmlNode` или объект `XmlNodeList`, в котором содержится узел или узлы, соответствующие заданным критериям поиска.

3.1. Метод GetElementsByTagName()

Этот метод реализован для классов `XmlDocument` и `XmlElement`. Он возвращает `XmlNodeList` (семейство объектов `XmlNode`), указывающий на все элементы, у которых имя тега совпадает с заданным и которые являются дочерними элементами определенного `XmlDocument` или `XmlElement`. В качестве примера воспользуемся XML-файлом `customerList.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<customerList>
  <customer customerID="12345">
    <name>Fred Q. Somebody</name>
    <address>100 Somewhere Street</address>
    <city>Sometown</city>
    <state>XY</state>
    <postalCode>13579</postalCode>
  </customer>
  <customer customerID="23456">
    <name>Jimmy C. Anybody</name>
    <address>200 Anywhere Lane</address>
    <city>Anytown</city>
    <state>YZ</state>
    <postalCode>99999</postalCode>
  </customer>
  <customer customerID="34567">
    <name>Mark Z. Somebody</name>
    <address>300 Nowhere Avenue</address>
    <city>Nowheresville</city>
    <state>XY</state>
    <postalCode>94949</postalCode>
  </customer>
</customerList>
```

Следующая программа позволяет получить сначала все элементы `<city>`, имеющиеся в документе, а затем элементы `<city>`, которые находятся в первом элементе `<customer>`:

```
using System;
using System.Xml;
using System.IO;
```

```
namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("customerList.xml", FileMode.Open);
            myDocument.Load(myStream);
            Console.Write("City elements found in the entire document:\r\n");
            XmlNodeList myCities = myDocument.GetElementsByTagName("city");
            for (int i = 0; i < myCities.Count; i++)
                Console.Write(myCities[i].ChildNodes[0].Value + "\r\n");
            Console.Write("City elements found in the first customer element:\r\n");
            XmlElement firstCustomer = (XmlElement)myDocument.DocumentElement.ChildNodes[0];
            myCities = firstCustomer.GetElementsByTagName("city");
            for (int i = 0; i < myCities.Count; i++)
                Console.Write(myCities[i].ChildNodes[0].Value + "\r\n");
            myStream.Close();
            Console.ReadLine();
        }
    }
}
```

Программа выдаст следующий результат:

```
City elements found in the entire document:
Sometown
Anytown
Nowheresville
City elements found in the first customer element:
Sometown
```

3.2. Метод GetElementsById()

Этот метод основан на использовании объекта `XmlDocument`. Он возвращает элемент с атрибутом типа ID, который совпадает с заданным ID. В данном случае необходимо, чтобы документ имел ассоциированную с ним схему или DTD – реализация DOM не основывается на предположении о том, что атрибут обладает ID, если он явно не описан в DTD или схеме. Для демонстрации этого метода воспользуемся XML-файлом `parts.xml`, внутри которого содержится DTD:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE partList [
  <!ELEMENT partList (part+)>
  <!ELEMENT part (name, size, color)>
  <!ATTLIST part partID ID #REQUIRED>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT size (#PCDATA)>
  <!ELEMENT color (#PCDATA)>
]>

<partList>
  <part partID="part17">
    <name>Grommets</name>
    <size>2 inch</size>
    <color>Red</color>
  </part>
  <part partID="part22">
    <name>Widgets</name>
    <size>3 inch</size>
    <color>Blue</color>
  </part>
```

```
</partList>
```

Следующая программа позволяет получить такой элемент <part>, у которого <partID> имеет значение part17:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("parts.xml", FileMode.Open);
            XmlValidatingReader myReader = new XmlValidatingReader(myStream, XmlNodeType.Document,
null);
            myDocument.Load(myReader);
            Console.Write("Part name for part ID part17:\r\n");
            XmlElement part17 = myDocument.GetElementById("part17");
            Console.Write(part17.ChildNodes[0].ChildNodes[0].Value + "\r\n");
            myStream.Close();
            Console.ReadLine();
        }
    }
}
```

Предупреждение.

```
"System.Xml.XmlValidatingReader" является устаревшим: "Use XmlReader created by
XmlReader.Create() method using appropriate XmlReaderSettings instead.
```

После выполнения этой программы будет получен следующий результат:

```
Part name for part ID part17:
Grommets
```

В данной программе объект XmlDocument пришлось загружать, используя объект XmlValidatingReader, - иначе процессор не смог бы распознать то, что атрибут partID является на самом деле ID, и в этом случае метод GetElementById() не сработал бы. Вопрос использования XmlValidatingReader для проверки допустимости DTD и схем обсуждается в ЛР № 5.

3.3. Метод SelectNodes()

Этот метод позволяет выбирать все узлы, которые относятся к XmlNode и соответствуют заданному выражению Xpath, и возвращает их в XmlNodeList. Этот метод является чрезвычайно мощным средством быстрого доступа к содержимому XML-документа в том случае, если указатели на запрашиваемые узлы оказываются более сложными, чем простое имя или ID элемента. Например, применительно к XML-файлу customerList.xml следующая программа позволяет извлекать значение элемента <city> для всех клиентов, чье значение state равно XY:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
```

```
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("customerList.xml", FileMode.Open);
            myDocument.Load(myStream);
            Console.Write("Cities for customers that are in the XY state:\r\n");
            XmlNodeList XYCities =
myDocument.SelectNodes("//customer/city/text()[../../state/text()='XY']");
            for (int i = 0; i < XYCities.Count; i++)
                Console.Write(XYCities[i].Value + "\r\n");
            myStream.Close();
            Console.ReadLine();
        }
    }
}
```

После выполнения этой программы будет получен следующий результат:

```
Cities for customers that are in the XY state:
Sometown
Nowheresville
```

3.4. Метод SelectSingleNode()

`SelectSingleNode()`, так же как и `SelectNodes()`, позволяет производить поиск по документу, используя XPath-выражение. Однако в этом случае метод будет возвращать только первый узел относительно начального узла (в порядке их расположения в документе), который удовлетворяет заданному условию поиска. Например, применительно к XML-файлу `customerList.xml` следующая программа будет извлекать название города только для первого клиента, чье значение `state` равно `XY`:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("customerList.xml", FileMode.Open);
            myDocument.Load(myStream);
            Console.Write("City for the first customer that is in the XY state:\r\n");
            XmlNode XYCity =
myDocument.SelectSingleNode("//customer/city/text()[../../state/text()='XY']");
            Console.Write(XYCity.Value + "\r\n");
            myStream.Close();
            Console.ReadLine();
        }
    }
}
```

После выполнения этой программы будет получен следующий результат:

```
City for the first customer that is in the XY state:
Sometown
```

4. Доступ к содержимому узлов

Рассмотрим ряд классов, производных от класса `XmlNode`, и выясним, как различные свойства этих классов

соотносятся с информацией, хранящейся в XML-документе, и каким образом можно извлекать эту информацию.

4.1. Работа с элементами

Выше уже встречались примеры, в которых осуществлялся доступ к узлам `XmlElement`. Имя элемента находится в свойстве `Name` объекта `XmlElement`, а содержимое – в семействе `ChildNodes`. Если необходимо получить доступ к атрибутам элемента, можно использовать метод `GetAttribute()`. В следующем примере возвращается текстовое содержимое элемента `<city>` для заказчика из следующего документа `customer.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<customer customerID="12345">
  <name>Fred Q. Somebody</name>
  <address>100 Somewhere Street</address>
  <city>Sometown</city>
  <state>XY</state>
  <postalCode>13579</postalCode>
</customer>
```

Программа, извлекающая содержимое элемента `<city>`, имеет вид:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("customer.xml", FileMode.Open);
            myDocument.Load(myStream);
            Console.Write("The customer's city:\r\n");
            XmlElement cityElement = (XmlElement)myDocument.DocumentElement.ChildNodes[2];
            Console.Write(cityElement.ChildNodes[0].Value + "\r\n");
            myStream.Close();
            Console.ReadLine();
        }
    }
}
```

После выполнения этой программы будет получен следующий результат:

```
The customer's city:
Sometown
```

Чтобы извлечь текстовое содержимое этого элемента за один вызов, можно воспользоваться свойством `InnerText` объекта `XmlElement`. Данный способ позволяет отделять ненужную разметку в смешанных элементах, например разметку параграфов. Так для XML-файла `para.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<para paraID="p17">
  This is a test of the <b>Emergency Broadcast System</b>.
</para>
```

программа:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("para.xml", FileMode.Open);
            myDocument.Load(myStream);
            Console.Write(myDocument.DocumentElement.InnerText + "\r\n");
            myStream.Close();
            Console.ReadLine();
        }
    }
}
```

выдаст следующий результат:

```
This is a test of the Emergency Broadcast System.
```

4.2. Работа с текстом

Для узлов `XmlText` интерес представляет свойство `Value`, в котором хранится собственно текстовое содержимое узла. Пример из предыдущего раздела демонстрирует, каким образом это свойство может быть использовано для извлечения значения узла `XmlText`. Если, однако, текст заключен в оболочку CDATA (в том случае, если он, например, содержит в себе разметочные символы), то анализатором будет создан узел `XmlCDATASection`. При наличии обычного текста и текста, заключенного в оболочку CDATA, каждая разновидность будет представлена отдельным узлом. Например, для XML-файла `sampleCode.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<sampleCode>
  <![CDATA[<?xml version="1.0"?>]]>
</sampleCode>
```

программа:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("sampleCode.xml", FileMode.Open);
            myDocument.Load(myStream);
            Console.Write(myDocument.DocumentElement.ChildNodes[0].Value + "\r\n");
            myStream.Close();
            Console.ReadLine();
        }
    }
}
```

выдаст следующий результат:

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

4.3. Работа с комментариями

Как и в случае узлов `XmlText`, содержимое узлов `XmlComment` может быть представлено с помощью свойства `Value` – оно будет содержать весь текст, расположенный между маркерами начала (`<!--`) и конца (`-->`) комментария. Например, для XML-файла `commentCode.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<commentCode>
  <!-- This is a useful comment about the sample document. -->
</commentCode>
```

можно воспользоваться той же самой программой, что и в предыдущем разделе, поскольку все, что нас интересует, – это значение `Value` первого дочернего узла документа. Результатом выполнения программы будет:

```
This is a useful comment about the sample document.
```

Т. к. не существует никаких гарантий того, что другие анализаторы не будут игнорировать комментарии, следует соблюдать аккуратность и не размещать в комментариях информацию, определяющую программируемое поведение.

4.4. Работа с инструкциями обработки

Инструкции обработки - это специальные инструкции, предназначенные для XML-анализатора. Их можно включать внутрь документа, и они могут быть прочитаны с помощью DOM-объектов. Существуют два различных способа получения доступа к содержимому узла `XmlProcessingInstruction`. Если требуется извлечь объект инструкции обработки и остальную часть инструкции по отдельности, то это можно осуществить с помощью свойств `Name` и `Data` объекта `XmlProcessingInstruction`. Например, для XML-файла `InstructionCode.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<InstructionCode>
  <?refreshData targetID="t17"?>
</InstructionCode>
```

программа:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("InstructionCode", FileMode.Open);
            myDocument.Load(myStream);
            XmlProcessingInstruction myPI = (XmlProcessingInstruction)
            myDocument.DocumentElement.ChildNodes[0];
            Console.Write("Name: " + myPI.Name + "\r\n");
            Console.Write("Data: " + myPI.Data + "\r\n");
            myStream.Close();
            Console.ReadLine();
        }
    }
```

```
}
```

выдаст следующий результат:

```
Name: refreshData
Data: targetID="t17"
```

4.5. Работа с атрибутами

Работа с атрибутами в DOM-модели выполняется иначе, нежели работа с узлами других типов. Поскольку атрибуты нечувствительны к порядку расположения (т. е. не имеет значения, в какой именно последовательности они указаны в открывающем теге элемента), работа с ними ведется как с неупорядоченным семейством узлов, связанных с конкретным узлом `XmlElement`, а не как с дочерними узлами данного узла. Если требуется получить доступ к списку всех узлов-атрибутов, связанных с конкретным `XmlElement`, можно воспользоваться свойством `Attributes` объекта `XmlElement`. Оно возвращает семейство атрибутов, которое можно просмотреть в цикле, либо можно организовать поиск по имени. Доступ к каждому объекту `XmlAttribute` в семействе осуществляется посредством свойств `Name` и `Value` объекта. Для примера возьмем XML-файл `attributes.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<customer customerID="12345"
          name="Fred Q. Somebody"
          address="100 Somewhere Street"
          city="Sometown"
          state="XY"
          postalCode="13579" />
```

Следующая программа возвращает имена и значения всех атрибутов элемента `<customer>`:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("attributes.xml", FileMode.Open);
            myDocument.Load(myStream);
            XmlAttributeCollection myAttributes = myDocument.DocumentElement.Attributes;
            for (int i = 0; i < myAttributes.Count; i++)
                Console.Write("Attribute: " + myAttributes[i].Name + " = " + myAttributes[i].Value +
"\r\n");
            myStream.Close();
            Console.ReadLine();
        }
    }
}
```

В итоге будет получен следующий результат:

```
Attribute: customerID = 12345
Attribute: name = Fred Q. Somebody
Attribute: address = 100 Somewhere Street
Attribute: city = Sometown
Attribute: state = XY
Attribute: postalCode = 13579
```

В качестве альтернативы для определения значения атрибута можно использовать один из вспомогательных методов `XmlElement: GetAttribute()` или `GetAttributeNode()`.

Метод `GetAttribute()` возвращает значение атрибута в текстовом виде по заданному имени для рассматриваемого элемента. Если использовать XML-файл `attributes.xml` и приведенную ниже программу:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("attributes.xml", FileMode.Open);
            myDocument.Load(myStream);
            Console.Write("The city attribute is: " +
myDocument.DocumentElement.GetAttribute("city"));
            myStream.Close();
            Console.ReadLine();
        }
    }
}
```

то будет получен следующий результат:

```
The city attribute is: Sometown
```

Метод `GetAttributeNode()` возвращает объект `XmlAttribute`, представляющий нужный атрибут, а не значение атрибута. Это оказывается полезным в том случае, когда значение атрибута включает в себя неопределенные ссылки на сущности или другое сложное содержимое, например:

```
<customer pref="&FedEx;-&Prioriti;" />
```

5. Работа с сущностями

Объект XmlDocument обычно не занимается проверкой допустимости, тем не менее он извлекает значения сущностей, объявленных во внутренних или во внешних подмножествах DTD (или схемах), и заменяет их в тех местах XML-документа, где встречает их объявление. Если ему не удается распознать объявление сущности, выводится сообщение об ошибке (даже в том случае, если не производится никаких других проверок содержимого на допустимость). Хотя и существует класс `XmlEntityReference`, он обычно применяется при создании XML-документов с помощью класса `XmlDocument`, и не появляется в анализируемых XML-документах.

5.1. Работа с пробелами

Объект `XmlDocument` обладает свойством `PreserveWhitespace`, которое управляет тем, как DOM-анализатор будет обрабатывать пробелы (символы табуляции, обычные пробелы, возвраты каретки и т.д.). Оно может исполнять две разные роли в зависимости от своего значения в процессе обращений к методу `Load()` (или `LoadXml()`) и к методу `Save()`. Если свойство `PreserveWhitespace` объекта `XmlDocument` имеет значение `true` в момент вызова метода `Load()` (или `LoadXml()`), то все узлы пробелов сохраняются. Реализация DOM в .NET может работать с двумя особыми типами узлов: `XmlWhitespace` и `XmlSignificantWhitespace`, с помощью которых представляются пробелы в исходном документе. Пробел в XML-документе считается значимым, если он располагается между двумя узлами, не являющимися

узлами пробелов. Под действие этого правила не подпадают единичные пробелы. Такие узлы войдут в создаваемую DOM-модель в том случае, если свойство `PreserveWhitespace` имеет значение `true`, и не войдут в противном случае. Значение свойства `PreserveWhitespace` по умолчанию – `false`. В качестве примера рассмотрим XML-файл `para2.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<para paraID="p17">
  This is a test of the <b>Emergency Broadcast System.</b>
  <i>This sentence is entirely in italics.</i>
</para>
```

Следующая программа не сохранит узлы пробелов:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("para2.xml", FileMode.Open);
            myDocument.Load(myStream);
            for (int i = 0; i < myDocument.DocumentElement.ChildNodes.Count; i++)
            {
                XmlNode myNode = myDocument.DocumentElement.ChildNodes[i];
                Console.Write("Node type: " + myNode.NodeType.ToString() + "\r\n");
            }
            myStream.Close();
            Console.ReadLine();
        }
    }
}
```

Выполнение программы приведет к получению следующего результата:

```
Node type: Text
Node type: Element
Node type: Element
```

А следующая программа сохранит пробелы при загрузке документа:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("para2.xml", FileMode.Open);
            myDocument.PreserveWhitespace = true;
            myDocument.Load(myStream);
            for (int i = 0; i < myDocument.DocumentElement.ChildNodes.Count; i++)
            {
                XmlNode myNode = myDocument.DocumentElement.ChildNodes[i];
                Console.Write("Node type: " + myNode.NodeType.ToString() + "\r\n");
```

```
            }
            myStream.Close();
            Console.ReadLine();
        }
    }
}
```

Выполнение программы приведет к получению следующего результата:

```
Node type: Text
Node type: Element
Node type: Whitespace
Node type: Element
Node type: Whitespace
```

Обычно свойству `PreserveWhitespace` следует присваивать значение `true` в случае загрузки документа, имеющего структуру текста (когда пробелы могут оказаться весьма важными), и оставлять значение `false` при загрузке документа, имеющего структуру данных. Если при вызове метода `Save()` флаг `PreserveWhitespace` установлен в значение `true`, все узлы пробелов в исходном дереве (узлы типа `XmlWhitespace` или `XmlSignificantWhitespace`) переносятся в результат, при этом никакого дополнительного форматирования не производится. Однако если свойство `PreserveWhitespace` имеет значение `false`, все узлы пробелов в исходном дереве отбрасываются, и метод `Save()` автоматически выполняет форматирование результата.

6. Поддержка пространств имен

Пространство имен – это средство, которое позволяет идентифицировать элементы как принадлежащие к определенным группам. Достигается это за счет объявления пространства имен и использования идентификатора пространства имен в соответствующих узлах в качестве префикса. Нередко эти имена применяются в качестве дополнительных ключей для процессора, которые указывают, каким образом следует использовать информацию. Например, пространство имен XSLT служит для сообщения процессору о том, что элементы, обладающие данным префиксом, представляют собой XSLT-инструкции. Реализация DOM, применяемая в.NET, обеспечивает полную поддержку пространств имен. Абстрактный класс `XmlNode` обладает двумя свойствами - `NamespaceURI` и `Prefix`, которые совместно определяют пространство имен для конкретного узла. Например, для XML-файла `customer2.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<cust:customer xmlns:cust="http://localhost/customer">
  <cust:name>Fred Q. Anybody</cust:name>
</cust:customer>
```

следующая программа вернет информацию о пространстве имен, применяемом в корневом элементе:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("customer2.xml", FileMode.Open);
            myDocument.Load(myStream);
            Console.Write("Namespace: " + myDocument.DocumentElement.NamespaceURI + "\r\n");
            Console.Write("Prefix:    " + myDocument.DocumentElement.Prefix + "\r\n");
            Console.Write("LocalName: " + myDocument.DocumentElement.LocalName + "\r\n");
            myStream.Close();
```

```
            Console.ReadLine();
        }
    }
}
```

Эта программа выдаст следующий результат:

```
Namespace: http://localhost/customer
Prefix:    cust
LocalName: customer
```

В данной программе для получения имени элемента использовано свойство `LocalName`. Если бы для получения имени элемента, имеющего префикс пространства имен, использовалось свойство `Name`, то было бы выдано все имя целиком (включая префикс и двоеточие). Эти свойства доступны для любого класса, являющегося производным от класса `XmlNode`. Однако для тех классов, которые не поддерживают пространства имен (таких, как `XmlText`), использование этих свойств не даст никакого эффекта.

7. Проверка допустимости

Сама по себе реализация DOM не выполняет проверку допустимости с применением DTD или схем XSD для документов, которые загружаются с помощью методов `Load()` и `LoadXml()`. Если требуется выполнить проверку допустимости документа, то необходимо сначала загрузить документ, воспользовавшись объектом `XmlValidatingReader`, а затем передать этот объект конструктору объекта `XmlDocument`. Более подробно применение `XmlValidatingReader` для проверки допустимости документов в соответствии с DTD или схемами XSD рассматривалось в ЛР № 5.

8. Внесение изменений в документ

Достоинство DOM-модели XML-документа заключается в том, что она позволяет добавлять и удалять содержимое в любой точке дерева XML-документа.

8.1. Удаление содержимого

Если подлежащее удалению содержимое не является атрибутом, то для удаления отдельных узлов можно использовать метод `RemoveChild()` для узла-родителя данного узла. Допустим, что требуется удалить элемент `<postalCode>` из элемента `<customer>` в XML-файле `customer3.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<customer>
  <name>Fred Q. Anybody</name>
  <address>1 Anywhere Road</address>
  <city>Anytown</city>
  <state>XY</state>
  <postalCode>58757</postalCode>
</customer>
```

Это может выполнить следующая программа:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
```

```
            FileStream myStream = new FileStream("customer3.xml", FileMode.Open);
            myDocument.Load(myStream);
            XmlElement pcElement = (XmlElement)myDocument.GetElementsByTagName("postalCode")[0];
            myDocument.DocumentElement.RemoveChild(pcElement);
            myDocument.Save("customer4.xml");
            myStream.Close();
        }
    }
}
```

В результате будет получен новый XML-файл (в данном случае - `customer4.xml`):

```
<?xml version="1.0" encoding="utf-8"?>
<customer>
  <name>Fred Q. Anybody</name>
  <address>1 Anywhere Road</address>
  <city>Anytown</city>
  <state>XY</state>
</customer>
```

Если требуется удалить из элемента атрибут, то надо иметь в виду следующее. Для узла `XmlElement` существует несколько вспомогательных методов, которые позволяют удалить атрибут из списка атрибутов конкретного элемента. Наиболее полезным из них является метод `RemoveAttribute()`, который удаляет атрибут с заданным именем. Для примера рассмотрим XML-файл `customer5.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<customer name="Fred Q. Anybody"
          address="1 Anywhere Road"
          city="Anytown"
          state="XY"
          postalCode="58757" />
```

Для удаления атрибута `<postalCode>` из элемента `<customer>` можно воспользоваться следующей программой:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("customer5.xml", FileMode.Open);
            myDocument.Load(myStream);
            myDocument.DocumentElement.RemoveAttribute("postalCode");
            myDocument.Save("customer6.xml");
            myStream.Close();
        }
    }
}
```

В результате будет получен новый XML-файл (в данном случае – `customer6.xml`):

```
<?xml version="1.0" encoding="utf-8"?>
<customer name="Fred Q. Anybody" address="1 Anywhere Road" city="Anytown" state="XY" />
```

8.2. Внесение изменений в содержимое

Внесение изменений в существующие узлы XML-документа несколько сложнее. Поскольку в модели `XmlDocument` узлы доступны только для чтения, то, например, для переименования элемента потребуется создать новый узел, скопировать все относящиеся к нему дочерние узлы, а затем заменить старый узел вновь созданным в дереве документа. Однако изменение значений узлов выполняется более простым способом. Для примера возьмем XML-файл `customerList2.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<customerList>
  <customer>
    <name>Fred Q. Anybody</name>
    <address>1 Anywhere Road</address>
    <city>Anytown</city>
    <state>XY</state>
    <postalCode>18743</postalCode>
  </customer>
  <customer>
    <name>John X. Somebody</name>
    <address>2 Somewhere Street</address>
    <city>Someville</city>
    <state>YZ</state>
    <postalCode>99999</postalCode>
  </customer>
</customerList>
```

Следующая программа осуществляет замену всех элементов `<postalCode>`, приводя их в соответствие полному американскому стандарту Zip+4:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("customerlist2.xml", FileMode.Open);
            myDocument.Load(myStream);
            XmlNodeList zipValues = myDocument.SelectNodes("//customer/postalCode/text()");
            for (int i = 0; i < zipValues.Count; i++)
                zipValues[i].Value = zipValues[i].Value + "-0000";
            myDocument.Save("customerlist3.xml");
            myStream.Close();
        }
    }
}
```

В результате будет получен новый XML-файл (в данном случае – `customerList3.xml`):

```
<?xml version="1.0" encoding="utf-8"?>
<customerList>
  <customer>
    <name>Fred Q. Anybody</name>
    <address>1 Anywhere Road</address>
    <city>Anytown</city>
    <state>XY</state>
    <postalCode>18743-0000</postalCode>
  </customer>
```

```
  <customer>
    <name>John X. Somebody</name>
    <address>2 Somewhere Street</address>
    <city>Someville</city>
    <state>YZ</state>
    <postalCode>99999-0000</postalCode>
  </customer>
</customerList>
```

8.3. Создание нового содержимого

Конструкторы различных классов `XmlNode` (за очевидным исключением `XmlDocument`) являются защищенными - нельзя непосредственно создавать новые экземпляры этих классов, поскольку они должны привязываться к конкретному документу. Напротив, для создания новых узлов следует использовать методы, предоставляемые объектом `XmlDocument`. Он предлагает метод для всех производных `XmlNode`. В качестве примера рассмотрим XML-файл `part2.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<part>
  <name>Grommets</name>
  <size>2 in.</size>
</part>
```

Следующая программа создает элемент `<color>` и добавляет его в конец списка дочерних элементов элемента `<part>`:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("part2.xml", FileMode.Open);
            myDocument.Load(myStream);
            XmlElement colorElement = myDocument.CreateElement("color");
            XmlText colorText = myDocument.CreateTextNode("red");
            colorElement.AppendChild(colorText);
            myDocument.DocumentElement.AppendChild(colorElement);
            myDocument.Save("part3.xml");
            myStream.Close();
        }
    }
}
```

В результате будет получен новый XML-файл (в данном случае – `part3.xml`):

```
<?xml version="1.0" encoding="utf-8"?>
<part>
  <name>Grommets</name>
  <size>2 in.</size>
  <color>red</color>
</part>
```

8.4. Вставка содержимого

Для вставки узла, не являющегося атрибутом, в содержимое другого узла, можно воспользоваться одним из

методов, наследуемых от XmlNode. Подобный пример приводился выше (вставка текстового узла в элементный узел, а также вставка одного элементного узла в другой элементный узел). Существуют и другие методы, которые позволяют точно выбирать, в каком именно месте списка дочерних элементов данного элемента должен оказаться добавляемый элемент. Метод PrependChild() вставляет дочерний узел в самое начало списка дочерних узлов узла-родителя, а методы InsertBefore() (перед) и InsertAfter() (после) позволяют выбрать существующий узел, относительно которого будет определяться местоположение нового узла.

8.5. Добавление атрибутов

Процедура добавления атрибутов в уже существующий элемент такова. Можно либо непосредственно выполнять манипуляции над семейством XmlAttributeCollection, либо воспользоваться одной из вспомогательных функций самого объекта XmlElement. Наиболее ценной является функция SetAttribute(), которая позволяет установить в указанное значение конкретный атрибут данного элемента. Например, для XML-файла part2.xml следующая программа добавляет атрибут partID в элемент <part>:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            FileStream myStream = new FileStream("part2.xml", FileMode.Open);
            myDocument.Load(myStream);
            myDocument.DocumentElement.SetAttribute("partID", "p17");
            myDocument.Save("part5.xml");
            myStream.Close();
        }
    }
}
```

В результате будет получен новый XML-файл (в данном случае – part5.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<part partID="p17">
  <name>Grommets</name>
  <size>2 in.</size>
</part>
```

9. Создание нового документа

Для создания нового документа следует воспользоваться классом XmlDocument. Сначала вызывается конструктор XmlDocument без параметров – в этот момент XmlDocument пуст. Затем с помощью метода CreateElement() создается корневой элемент, который объявляется корневым элементом документа посредством вызова метода AppendChild() для объекта XmlDocument. После этого создаются элементы, атрибуты и т. п. по мере необходимости и строится дерево документа посредством вызова соответствующих методов типа AppendChild(). Допустим, что нужно создать следующий XML-файл part6.xml:

```
<part partID="p17">
  <name>Grommets</name>
  <size>2 in.</size>
</part>
```

Это можно сделать с помощью программы:

```
using System;
using System.Xml;
using System.IO;

namespace Wrox
{
    class consoleApp
    {
        static void Main(string[] args)
        {
            XmlDocument myDocument = new XmlDocument();
            XmlElement partElement = myDocument.CreateElement("part");
            partElement.SetAttribute("partID", "p17");
            myDocument.AppendChild(partElement);
            XmlElement nameElement = myDocument.CreateElement("name");
            XmlText nameText = myDocument.CreateTextNode("Grommets");
            nameElement.AppendChild(nameText);
            partElement.AppendChild(nameElement);
            XmlElement sizeElement = myDocument.CreateElement("size");
            XmlText sizeText = myDocument.CreateTextNode("2 in.");
            sizeElement.AppendChild(sizeText);
            partElement.AppendChild(sizeElement);
            myDocument.Save("part6.xml");
        }
    }
}
```

Следует помнить, что документ будет представлен в памяти исключительно в виде DOM-модели до тех пор, пока не будет вызван метод типа `Save()`, который позволит сериализовать документ для хранения в файле или на каком-либо другом носителе.

Задание

Написать консольное приложение на языке C#, которое для XML-документа, полученного в ЛР № 5, выполняет следующие функции:

1. Открытие документа, находящегося в файле.
2. Поиск информации, содержащейся в документе:
    a. с помощью метода GetElementsByTagName,
    b. с помощью метода GetElementsById,
    c. с помощью метода SelectNodes,
    d. с помощью метода SelectSingleNode.
3. Доступ к содержимому узлов:
    a. к узлам типа XmlElement,
    b. к узлам типа XmlText,
    c. к узлам типа XmlComment,
    d. к узлам типа XmlProcessingInstruction,
    e. к атрибутам узлов.
4. Внесение изменений в документ:
    a. удаление содержимого,
    b. внесение изменений в содержимое,
    c. создание нового содержимого,
    d. вставка содержимого,
    e. добавление атрибутов.

Замечания.

1. Результаты выполнения пунктов 2 и 3 выводить на консоль.

2. Результаты выполнения пункта 4 сохранять в файле.
3. Работой программы должно управлять консольное меню со структурой задания.