

ЛЕКЦИЯ 9 – част 2

9

СЪДЪРЖАНИЕ

I. Работа с XML

5. XML парсери, общи сведения, същност и видове

6. Стандартите DOM и SAX

7. XML и .NET Framework

I.5. XML парсери, общи сведения, същност и видове

Терминът **парсване (parse)** е **процес на взимането на фрагмент с код**, описан в синтаксиса на този език, **и разбиването му на отделни компоненти, дефинирани от езиковите правила** заложиени в основата на езика. На български език терминът парсване може да бъде заменен с „**синтактичен анализ**“.

XML парсерите са библиотеки, които четат XML документи, извличат от тях таговете и съдържаната в тях информация и ги предоставят за обработка на програмиста. Те предоставят и функционалност за построяване на нови и промяна на вече създадени XML документи.

Приложение на XML парсерите:

- извличане на данни от XML документи;
- построяване на нови XML документи;
- промяна на съществуващи XML документи;
- валидация на XML документи по зададена схема.

I. РАБОТА С XML



XML парсери – видове

XML парсерите могат да се класифицират по различни критерии. От една страна те се делят на:

- **валидиращи** - нуждаят се от DTD или XSD схема, по която да валидират документите;
- **невалидиращи** - изискват единствено добре дефинирани документи, които да обработват.

По начина на работа се разграничават:

- **дървовидно-ориентирани** – DOM (Document Object Model);
- **поточно-ориентирани** – SAX (Simple API for XML Processing).

I.6. Стандартите DOM и SAX

Ще започнем с DOM стандарта и ще разгледаме обектния модел, който той дефинира.

Какво представлява DOM стандарта ?

Документният обектен модел DOM (Document Object Model) дефинира платформено и езиково-независим програмен интерфейс за достъп и манипулиране на съдържанието и структурата на XML документите, като дървовидни структури в паметта. XML документният обектен модел е базиран на W3C DOM спецификацията и е утвърден световен стандарт.

Документният обектен модел представя един XML документ **като дървовидна йерархия от възли**. DOM стандартът дефинира следните типове възли: Document, Element, DocumentFragment, DocumentType, Attr, Text, EntityReference, ProcessingInstruction, Comment, CDATASection, Entity и Notation. Някои от тези типове могат да имат наследници, като за всеки възел те са определени в DOM спецификацията.

Документният обектен модел определя също типовете **NodeList** за обработка на колекции и **NamedNodeMap** за речникови обекти от тип ключ-стойност.

DOM спецификацията описва интерфейси, а не действителни класове и обекти и затова за работа с нея ни е необходима конкретна имплементация (DOM парсер).

Кога се използва DOM стандартът

DOM предоставя отлична функционалност в следните случаи:

- при обработката на обемисти XML документи;
- при нужда от произволен достъп до различни части от XML документа по различно време;
- За приложения, които променят структурата на XML документа "в движение«.

При DOM стандартът е необходимо първоначално целият документ да бъде прочетен и съхранен в паметта, което може да доведе до забавяне на обработката и излишно изразходване на изчислителни ресурси.

I. РАБОТА С XML



Стандартът SAX

SAX (Simple API for XML Processing) е базиран на събития, програмен интерфейс, който чете XML документи последователно като поток и позволява анализиране на съдържанието на тези документи.

Обработката на XML документи, базирана на събития, следи за наличието на ограничен брой събития, като:

- срещане на отварящи и затварящи тагове на елементи;
- character data;
- коментари;
- инструкции за обработка и др.

Всеки път, когато парсерът срещне отварящ или затварящ таг, character data или друго събитие, той известява за това програмата, която го използва.

I. РАБОТА С XML



Кога се използва стандартът SAX

При SAX базираните приложения XML документът се предоставя за обработка на програмата фрагмент по фрагмент от началото до края.

SAX приложението може да съхранява интересуващите го части, докато целият документ бъде прочетен, или може да обработва информацията в момента на получаването ѝ.

Не е нужно обработката на вече прочетени елементи да чака прочитането на целия документ и още по-важно – не е нужно целият документ да се съхранява в паметта, за да е възможна работата с него.

Тези характеристики правят SAX модела много удобен за обработка на обемисти XML документи, които не могат да бъдат заредени в паметта.

I.7. XML и .NET Framework

.NET Framework е проектиран от самото начало с идеята за силно интегрирана XML поддръжка.

Имплементациите на основните XML технологии се съдържат в асемблите System.Xml, където са дефинирани следните главни пространства от имена:

- **System.Xml** – осигурява основните входно-изходни операции с XML (XmlReader и XmlWriter) и други XML помощни класове.
- **System.Xml.Schema** – осигурява поддръжка на валидация на XML съдържание чрез XML Schema (XmlSchemaObject и наследниците му).
- **System.Xml.XPath** – реализира функционалност за XPath търсене на информация и навигация в XML документ (класовете XPathDocument, XPathNavigator и XPathExpression).

I. РАБОТА С XML

.NET и DOM моделът

.NET Framework предоставя пълен набор от класове, които зареждат и редактират XML документи. **Основният XML DOM клас в .NET Framework е XmlDocument.** Силно свързан с него е неговият клас-наследник **XMLDataDocument**, който разширява XmlDocument и акцентира върху съхраняването и извличането на структурирани таблични данни в XML.

При работа с XML DOM модела XML документът първо се зарежда целият като дърво в паметта и едва тогава се обработва. XML DOM предоставя средства за навигация и редактиране на XML документа и поддържа XPath заявки и XSL трансформации.

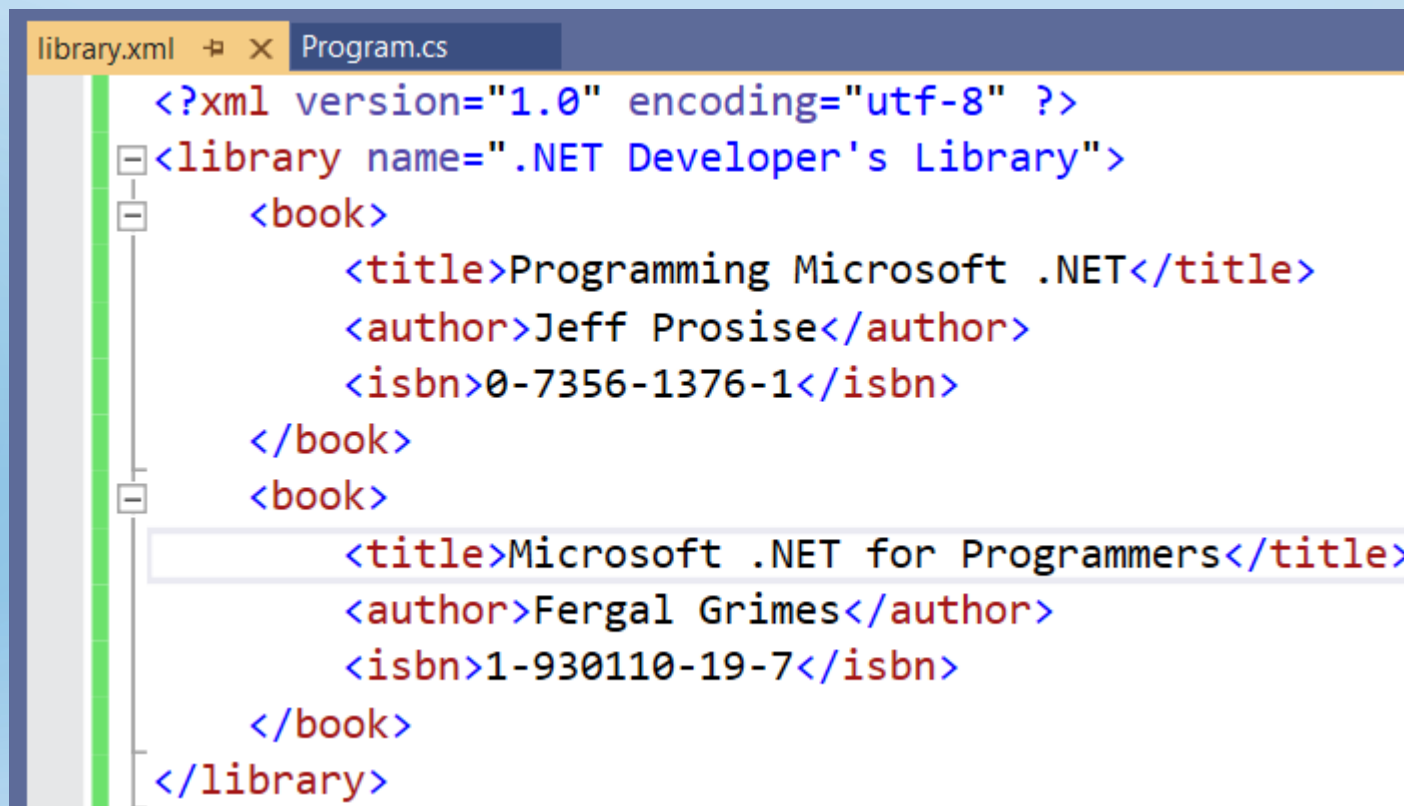
I. РАБОТА С XML

9

Парсване на XML документ с DOM – пример

Примерът илюстрира използването на DOM парсер за парсване на XML документ, обхождане на полученото DOM дърво и извличане на информация от него.

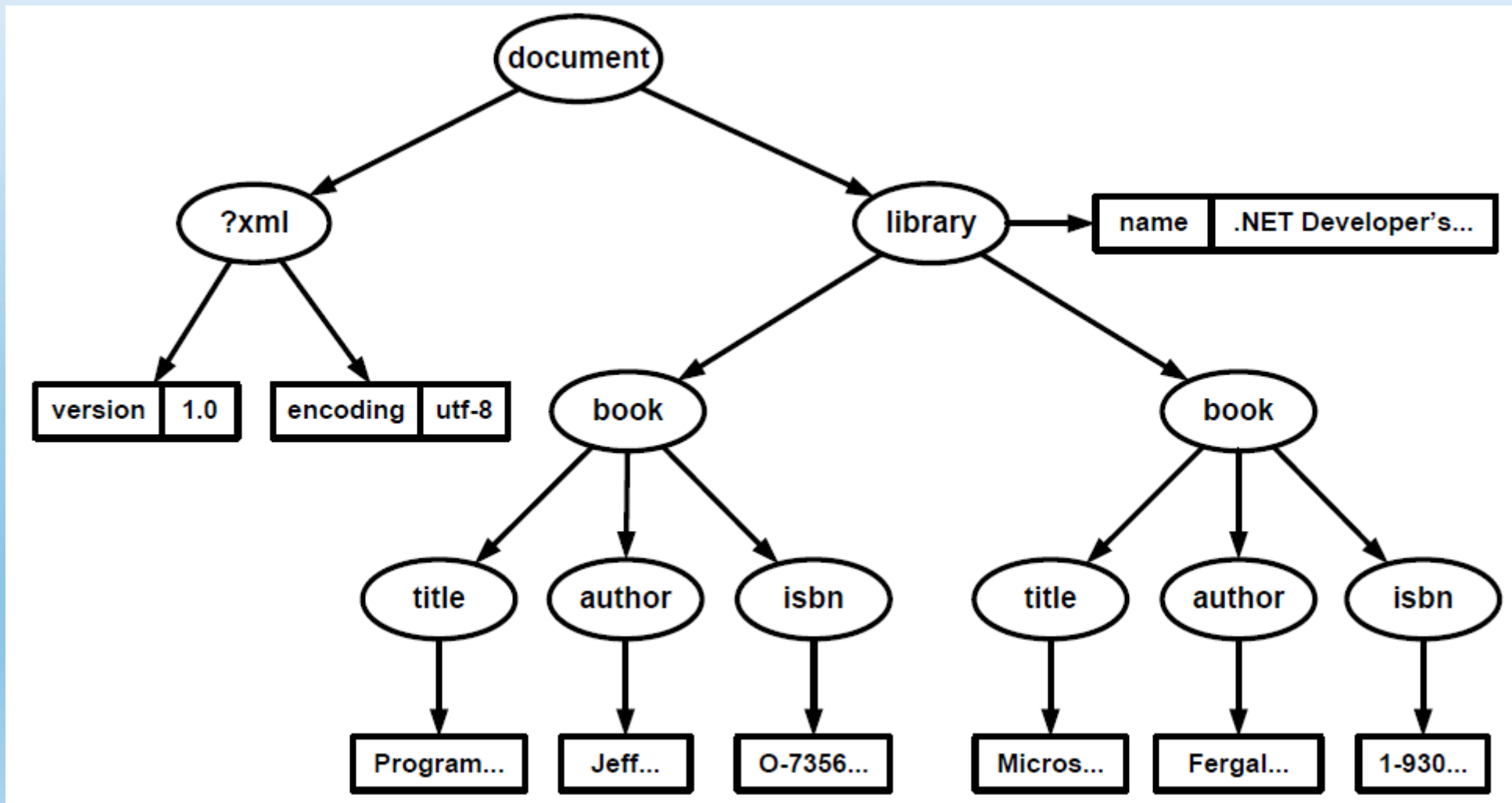
За целта ни е необходим работен XML документ (с име например **library.xml**):



```
library.xml Program.cs
<?xml version="1.0" encoding="utf-8" ?>
<library name=".NET Developer's Library">
  <book>
    <title>Programming Microsoft .NET</title>
    <author>Jeff Prosise</author>
    <isbn>0-7356-1376-1</isbn>
  </book>
  <book>
    <title>Microsoft .NET for Programmers</title>
    <author>Fergal Grimes</author>
    <isbn>1-930110-19-7</isbn>
  </book>
</library>
```

I. РАБОТА С XML

Този документ се представя в паметта като DOM дърво по следния начин:



I. РАБОТА С XML

9

Програмният пример, който ще разгледаме стъпка по стъпка (написан на езика C# в средата Visual studio има за цел да извлече книгите от файла **library.xml** и да отпечата информация за тях – заглавие, автор и ISBN.

1. Създаваме нов проект във VS.NET – конзолно приложение.

2. След като сме създали проект, първото, което е необходимо да направим, е да създадем XML файла **library.xml**, за да можем в последствие да го подложим на обработка. След зареждането на документа отпечатваме съдържанието му, за да се уверим, че зареждането е успешно. За отпечатване на информацията от файла използваме кода по-долу:

```
12 static void Main(string[] args)
13 {
14     XmlDocument doc = new XmlDocument();
15     doc.Load("library.xml");
16     Console.WriteLine("Loaded XML document:");
17     Console.WriteLine(doc.OuterXml);
18     Console.WriteLine();
19 }
20 }
21 }
```

I. РАБОТА С XML

3. Извличаме документния елемент на XML файла и отпечатваме името му на КОНЗОЛНИЯ ИЗХОД:

```
20 XmlNode rootNode = doc.DocumentElement;  
21 Console.WriteLine("Root node: {0}", rootNode.Name);
```

4. Обхождаме и отпечатваме атрибутите на документния елемент (в този случай имаме един единствен атрибут name):

```
26 foreach (XmlAttribute atr in rootNode.Attributes)  
27 {  
28     Console.WriteLine("Attribute: {0}={1}",  
29         atr.Name, atr.Value);  
30 }
```

I. РАБОТА С XML

5. Обхождаме всички елементи-деца на документния елемент. Всеки от тях описва една книга (елемент book). За всяка книга отпечатваме заглавието, автора и isbn номера, като ги извличаме от съответните им елементи, наследници на елемента book:

```
33      foreach (XmlNode node in rootNode.ChildNodes)
34      {
35          Console.WriteLine("Book title = {0}",
36                             node["title"].InnerText);
37          Console.WriteLine("Book author = {0}",
38                             node["author"].InnerText);
39          Console.WriteLine("Book isbn = {0}",
40                             node["isbn"].InnerText);
41          Console.WriteLine();
42      }
```


I. РАБОТА С XML



6. Резултат от изпълнението на цялата програма:

```
C:\Users\user\source\repos\ConsoleApp_UPR_10-XML\ConsoleApp_UPR_10-XML\bin\Debug\ConsoleApp_UPR_10-XML.exe
Loaded XML document:
<?xml version="1.0" encoding="utf-8"?><library name=".NET Developer's Library"><book><title>Programming Microsoft .NET</title><author>Jeff Prosise</author><isbn>0-7356-1376-1</isbn></book><book><title>Microsoft .NET for Programmers</title><author>Fergal Grimes</author><isbn>1-930110-19-7</isbn></book></library>

Root node: library
Attribute: name=.NET Developer's Library
Book title = Programming Microsoft .NET
Book author = Jeff Prosise
Book isbn = 0-7356-1376-1

Book title = Microsoft .NET for Programmers
Book author = Fergal Grimes
Book isbn = 1-930110-19-7
```

В примера използвахме класовете XmlDocument, XmlNode и XmlAttribute, от пространството System.Xml. Нека разгледаме за какво служат и как се използват тези класове.

I. РАБОТА С XML



Класовете за работа с DOM

Работата с DOM в .NET Framework се осъществява с помощта на следните по-важни класове:

- **XmlNode** – абстрактен базов клас за всички възли в едно DOM дърво;
- **XmlDocument** – съответства на корена на DOM дърво, обикновено съдържа два наследника: заглавна част (пролог) и документния елемент на XML документа;
- **XmlElement** – представя XML елемент;
- **XmlAttribute** – представя атрибут на XML елемент (двойка име-стойност);
- **XmlAttributeCollection** – списък от XML атрибути;
- **XmlNodeList** – списък от възли в DOM дърво;

I. РАБОТА С XML



Класът XmlNode – този клас е най-важния клас в обектния модел на .NET за работа с XML документи и е базов клас за различните DOM типове възли.

Класът XmlNode представя базов възел и е класът, наследяван от всички специфични DOM възли (XmlDocument, XmlElement, XmlAttribute и т.н.). Неговите свойства осигуряват достъп до вътрешните стойности на всеки възел:

- пространството от имена на възела;
- тип на възела;
- възел-родител;
- възел-наследник;
- съседни възли и др.

XmlNode позволява навигация в DOM дървото!

Класът XmlNode предоставя набор от средства за навигация чрез своите свойства:

- **ParentNode** – връща възела-родител (или null ако няма);
- **PreviousSibling / NextSibling** – връща левия / десния съсед на текущия възел;
- **FirstChild / LastChild** – връща първия / последния наследник на текущия възел;
- **Item** (индексатор в C#) – връща наследник на текущия възел по името му;

XmlNode позволява работа с текущия възел в DOM дървото чрез:

- **Name** – връща името на възела (име на елемент, атрибут, ...).
- **Value** – връща стойността на възела.



Стойността на свойството `Value` в голяма степен зависи от типа на конкретно разглеждания възел. За възел от тип атрибут това свойство наистина връща стойността му, но за възел от тип елемент например, `Value` връща нулева референция. Стойността на елементите се достъпва през свойствата `InnerText` и `InnerXml`. За пълен списък на връщаните от `Value` стойности за различните DOM възли потърсете в MSDN.

- **Attributes** – връща списък от атрибутите на възела (като `XmlAttributeCollection`);
- **HasChildNodes** – връща булева стойност дали има възелът има наследници;
- **InnerXml, OuterXml** – връща частта от XML документа, която описва съдържанието на възела съответно без и с него самия;
- **InnerText** – връща конкатенация от стойностите на възела и наследниците му рекурсивно;
- **NodeType** – връща типа на възела.

XmlNode позволява промяна на текущия възел:

- **AppendChild(...)** / **PrependChild(...)** – добавя нов наследник след / преди всички други наследници на текущия възел.
- **InsertBefore(...)** / **InsertAfter(...)** – вмъква нов наследник преди / след указан наследник.
- **RemoveChild(...)** / **ReplaceChild(...)** – премахва / заменя указания наследник.
- **RemoveAll()** – изтрива всички наследници на текущия възел (атрибути, елементи, ...).
- **Value, InnerText, InnerXml** – променя стойността / текста / XML текста на възела.

Класът XmlDocument - Класът XmlDocument съдържа един XML документ във вид на DOM дърво според W3C спецификацията за документния обектен модел. Документът е представен като дърво от възли, които съхраняват елементите, атрибутите и техните стойности и съдържат информация за родител, наследник и съседни възли.

Основни свойства, методи и събития в класа:

- **Load(...), LoadXml(...), Save(...)** – позволяват зареждане и съхранение на XML документи от и във файл, поток или символен низ
- **DocumentElement** – извлича документния елемент на XML документа.
- **PreserveWhitespace** – указва дали празното пространство да бъде запазено при зареждане / записване на документа.
- **CreateElement(...), CreateAttribute(...), CreateTextNode(...)** – създа-ва нов XML елемент, атрибут или стойност на елемент.
- **NodeChanged, NodeInserted, NodeRemoved** – събития за следене за промени в документа.

I. РАБОТА С XML



Промяна на XML документ с DOM – пример

Условие на задачата: Да се удвоят цените на бирата в даден XML документ, но същевременно да се запазят непроменени цените на останалите стоки в списъка.

За целта ще е необходимо да прочетем целия XML документ в паметта и да анализираме стоките една по една. При срещане на елемент, който идентифицираме като бира, удвояваме цената му, а в противен случай не предприемаме никакво действие.

РЕШЕНИЕ:

1. Създаваме нов проект във VS.NET – конзолно приложение.
2. Създаваме на нов XML файл с име **items.xml**.
3. Въвеждане на данните в XML файла.

I. РАБОТА С XML

9

```
items.xml  Program.cs
<?xml version="1.0" encoding="utf-8" ?>
<items>
  <item type="beer">
    <name>Загорка</name>
    <price>0.54</price>
  </item>
  <item type="food">
    <name>кебапчета</name>
    <price>0.48</price>
  </item>
  <item type="beer">
    <name>Каменица</name>
    <price>0.56</price>
  </item>
</items>
```

4. Зареждаме работния XML документ items.xml в паметта, за да го подготвим за предстоящата манипулация:

```
12 static void Main(string[] args)
13 {
14     XmlDocument doc = new XmlDocument();
15     doc.Load("items.xml");
16 }
```

5. Естеството на този пример ни задължава да работим с десетични числа. В XML документа те са форматираны с десетична точка, но винаги съществува вероятност текущата активна култура на компютъра, където изпълняваме програмата, да е различна и да форматира числата с десетична запетая (например българската култура). За да се подсигуриим, че парсването на числата ще протече безпроблемно и няма да предизвика изключение от тип `FormatException`, най-правилно е да използваме специалната културно-необвързана култура, достъпна през свойството **`CultureInfo.InvariantCulture`**.

I. РАБОТА С XML

9

5. Обхождаме наследниците `item` на документния елемент `items` и за всеки от тях, чийто атрибут `type` има стойност `"beer"`, прочитаме стойността на наследника му `price`. Дотук обаче имаме стойността на елемента `price` единствено като низ. Необходимо е да парснем низа към десетично число и именно тук използваме `CultureInfo.InvariantCulture`. Вече разполагаме с десетично число, което удвояваме и записваме на мястото на старата стойност на елемента `price` (отново е нужно да укажем културата, за да се предпазим от грешки). Ето как изглежда кода, който извършва манипулацията:

```
18      foreach (XmlNode node in doc.DocumentElement)
19      {
20          if (node.Attributes["type"].Value == "beer")
21          {
22              string currentPriceStr =
23                  node["price"].InnerText;
24              decimal currentPrice = Decimal.Parse(
25                  currentPriceStr, CultureInfo.InvariantCulture);
26              decimal newPrice = currentPrice * 2;
27              node["price"].InnerText =
28                  newPrice.ToString(CultureInfo.InvariantCulture);
29          }
30      }
```

I. РАБОТА С XML

9

6. Сега остава единствено да отпечатаме XML документа, за да се уверим, че промените действително са налице и след това да запазим промените в нов файл itemsNew.xml:

```
32 Console.WriteLine(doc.OuterXml);  
33 doc.Save(@"C:\Users\user\source\repos\ConsoleApp_UPR_10-XML-2\  
34 ConsoleApp_UPR_10-XML-2\itemsNew.xml");
```



The screenshot shows the XML document items.xml in a code editor. The XML is as follows:

```
<?xml version="1.0" encoding="utf-8" ?>  
<items>  
  <item type="beer">  
    <name>Загорка</name>  
    <price>0.54</price>  
  </item>  
  <item type="food">  
    <name>кебапчета</name>  
    <price>0.48</price>  
  </item>  
  <item type="beer">  
    <name>Каменица</name>  
    <price>0.56</price>  
  </item>  
</items>
```



The screenshot shows the XML document itemsNew.xml in a code editor. The XML is as follows:

```
<?xml version="1.0" encoding="utf-8"?>  
<items>  
  <item type="beer">  
    <name>Загорка</name>  
    <price>1.08</price>  
  </item>  
  <item type="food">  
    <name>кебапчета</name>  
    <price>0.48</price>  
  </item>  
  <item type="beer">  
    <name>Каменица</name>  
    <price>1.12</price>  
  </item>  
</items>
```


I. РАБОТА С XML



Построяване на XML документ с DOM – пример

Условие на задачата: Чрез използване на езика C# да бъде построен (генериран) XML файл със следното съдържание:

order.xml
<pre><order> <item ammount="4">бира</item> <item ammount="2">картофки</item> <item ammount="6">кебапчета</item> </order></pre>

РЕШЕНИЕ:

За целта трябва да създадем **XmlDocument**, да създадем и добавим документен елемент като негов наследник, след което да добавим към документния елемент още 3 елемента, като им зададем подходящо съдържание и им добавим по един атрибут за количество.

I. РАБОТА С XML

9

```
13 static void Main(string[] args)
14 {
15     XmlDocument xmlDoc = new XmlDocument();
16     XmlElement docElement = xmlDoc.CreateElement("order");
17     xmlDoc.AppendChild(docElement);
18     AppendItem(xmlDoc, docElement, "бира", 4);
19     AppendItem(xmlDoc, docElement, "картофки", 2);
20     AppendItem(xmlDoc, docElement, "кебапчета", 6);
21     xmlDoc.Save("order.xml");
22 }
```

```
24 static void AppendItem(XmlDocument aXmlDoc, XmlElement aXmlElement,
25                         string aItemName, int aAmmount)
26 {
27     XmlElement itemElement = aXmlDoc.CreateElement("item");
28     itemElement.InnerText = aItemName;
29     XmlAttribute ammountAttr =
30     aXmlDoc.CreateAttribute("ammount");
31     ammountAttr.Value = aAmmount.ToString();
32     itemElement.Attributes.Append(ammountAttr);
33     aXmlElement.AppendChild(itemElement);
34 }
```

order.xml Program.cs

```
<order>
  <item ammount="4">бира</item>
  <item ammount="2">картофки</item>
  <item ammount="6">кебапчета</item>
</order>
```