**XML И C#**

На сегодняшний день XML является одним из распространенных стандартов документов, который позволяет в удобной форме сохранять сложные по структуре данные. Поэтому разработчики платформы .NET включили в фреймворк широкие возможности для работы с XML.

Прежде чем перейти непосредственно к работе с XML-файлами, сначала рассмотрим, что представляет собой xml-документ и как он может хранить объекты, используемые в программе на c#.

Например, у нас есть следующий класс:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | class Person  {      public string Name { get;}      public int Age { get; set; }      public string Company { get; set; }      public Person(string name, int age, string company)      {          Name = name;          Age = age;          Company = company;      }  } |

В программе на C# мы можем создать список объектов класса Person:

|  |  |
| --- | --- |
| 1  2  3  4  5 | var employees = new List<Person>  {      new Person ("Tom", 37, "Microsoft"),      new Person ("Bob", 41, "Google")  }; |

Чтобы сохранить список в формате xml мы могли бы использовать следующий xml-файл:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?xml version="1.0" encoding="utf-8" ?>  <people>    <person name="Tom">      <company>Microsoft</company>      <age>37</age>    </person>    <person name="Bob">      <company>Google</company>      <age>41</age>    </person>  </people> |

XML-документ объявляет строка <?xml version="1.0" encoding="utf-8" ?>. Она задает версию (1.0) и кодировку (utf-8) xml. Далее идет собственно содержимое документа.

XML-документ должен иметь один единственный корневой элемент, внутрь которого помещаются все остальные элементы. В данном случае таким элементом является элемент <people>. Внутри корневого элемента <people> задан набор элементов <person>. Вне корневого элемента мы не можем разместить элементы person.

Каждый элемент определяется с помощью открывающего и закрывающего тегов, например, <person> и </person>, внутри которых помещается значение или содержимое элементов. Также элемент может иметь сокращенное объявление: <person /> - в конце элемента помещается слеш.

Элемент может иметь вложенные элементы и атрибуты. В данном случае каждый элемент person имеет два вложенных элемента company и age и атрибут name.

Атрибуты определяются в теле элемента и имеют следующую форму: название="значение". Например, <person name="Bill Gates">, в данном случае атрибут называется name и имеет значение Bill Gates

Внутри простых элементов помещается их значение. Например, <company>Google</company> - элемент company имеет значение Google.

Названия элементов являются регистрозависимыми, поэтому <company> и <COMPANY> будут представлять разные элементы.

Таким образом, весь список Users из кода C# сопоставляется с корневым элементом <people>, каждый объект Person - с элементом <person>, а каждое свойство объекта Person - с атрибутом или вложенным элементом элемента <person>

Что использовать для свойств - вложенные элементы или атрибуты? Это вопрос предпочтений - мы можем использовать как атрибуты, так и вложенные элементы. Так, в предыдущем примере вполне можно использовать вместо атрибута вложенный элемент:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <?xml version="1.0" encoding="utf-8" ?>  <people>    <person>      <name>Tom</name>      <company>Microsoft</company>      <age>37</age>    </person>    <person>      <name>Bob</name>      <company>Google</company>      <age>41</age>    </person>  </people> |

Для работы с XML в C# можно использовать несколько подходов. В первых версиях фреймворка основной функционал работы с XML предоставляло пространство имен **System.Xml**. В нем определен ряд классов, которые позволяют манипулировать xml-документом:

* **XmlNode**: представляет узел xml. В качестве узла может использоваться весь документ, так и отдельный элемент
* **XmlDocument**: представляет весь xml-документ
* **XmlElement**: представляет отдельный элемент. Наследуется от класса XmlNode
* **XmlAttribute**: представляет атрибут элемента
* **XmlText**: представляет значение элемента в виде текста, то есть тот текст, который находится в элементе между его открывающим и закрывающим тегами
* **XmlComment**: представляет комментарий в xml
* **XmlNodeList**: используется для работы со списком узлов

Ключевым классом, который позволяет манипулировать содержимым xml, является **XmlNode**, поэтому рассмотрим некоторые его основные методы и свойства:

* Свойство **Attributes** возвращает объект XmlAttributeCollection, который представляет коллекцию атрибутов
* Свойство **ChildNodes** возвращает коллекцию дочерних узлов для данного узла
* Свойство **HasChildNodes** возвращает true, если текущий узел имеет дочерние узлы
* Свойство **FirstChild** возвращает первый дочерний узел
* Свойство **LastChild** возвращает последний дочерний узел
* Свойство **InnerText** возвращает текстовое значение узла
* Свойство **InnerXml** возвращает всю внутреннюю разметку xml узла
* Свойство **Name** возвращает название узла. Например, <user> - значение свойства Name равно "user"
* Свойство **ParentNode** возвращает родительский узел у текущего узла

Применим эти классы и их функционал. И вначале для работы с xml создадим новый файл. Назовем его **people.xml** и определим в нем следующее содержание:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?xml version="1.0" encoding="utf-8" ?>  <people>    <person name="Tom">      <company>Microsoft</company>      <age>37</age>    </person>    <person name="Bob">      <company>Google</company>      <age>41</age>    </person>  </people> |

Теперь пройдемся по этому документу и выведем его данные на консоль:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | using System.Xml;    XmlDocument xDoc = new XmlDocument();  xDoc.Load("people.xml");  // получим корневой элемент  XmlElement? xRoot = xDoc.DocumentElement;  if (xRoot != null)  {      // обход всех узлов в корневом элементе      foreach (XmlElement xnode in xRoot)      {          // получаем атрибут name          XmlNode? attr = xnode.Attributes.GetNamedItem("name");          Console.WriteLine(attr?.Value);            // обходим все дочерние узлы элемента user          foreach (XmlNode childnode in xnode.ChildNodes)          {              // если узел - company              if (childnode.Name == "company")              {                  Console.WriteLine($"Company: {childnode.InnerText}");              }              // если узел age              if (childnode.Name == "age")              {                  Console.WriteLine($"Age: {childnode.InnerText}");              }          }          Console.WriteLine();      }  } |

**Загрузка документа: xDoc.Load("people.xml");**

При разборе xml для начала мы получаем корневой элемент документа с помощью свойства xDoc.DocumentElement. Далее уже происходит собственно разбор узлов документа.

В цикле foreach(XmlNode xnode in xRoot) пробегаемся по всем дочерним узлам корневого элемента. Так как дочерние узлы представляют элементы <person>, то мы можем получить их атрибуты: XmlNode attr = xnode.Attributes.GetNamedItem("name"); и вложенные элементы: foreach(XmlNode childnode in xnode.ChildNodes)

Чтобы определить, что за узел перед нами, мы можем сравнить его название: if(childnode.Name=="company")

Подобным образом мы можем создать объекты классов и структур по данным из xml:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | using System.Xml;    var people = new List<Person>();    XmlDocument xDoc = new XmlDocument();  xDoc.Load("people.xml");  // получим корневой элемент  XmlElement? xRoot = xDoc.DocumentElement;  if (xRoot != null)  {      foreach (XmlElement xnode in xRoot)      {          Person person = new Person();          XmlNode? attr = xnode.Attributes.GetNamedItem("name");          person.Name = attr?.Value;            foreach (XmlNode childnode in xnode.ChildNodes)          {              if (childnode.Name == "company")                  person.Company = childnode.InnerText;                if (childnode.Name == "age")                  person.Age = int.Parse(childnode.InnerText);          }          people.Add(person);      }      foreach (var person in people)          Console.WriteLine($"{person.Name} ({person.Company}) - {person.Age}");  }    class Person  {      public string? Name { get; set; }      public int Age { get; set; }      public string? Company { get; set; }  } |

В данном случае определен класс Person с тремя свойствами. При переборе узлов файла xml значения элементов и их атрибутов передается объекту класса Person.

**Изменение XML-документа**

Для редактирования xml-документа (изменения, добавления, удаления элементов) мы можем воспользоваться методами класса **XmlNode**:

* **AppendChild**: добавляет в конец текущего узла новый дочерний узел
* **InsertAfter**: добавляет новый узел после определенного узла
* **InsertBefore**: добавляет новый узел до определенного узла
* **RemoveAll**: удаляет все дочерние узлы текущего узла
* **RemoveChild**: удаляет у текущего узла один дочерний узел и возвращает его

Класс **XmlDocument** добавляет еще ряд методов, которые позволяют создавать новые узлы:

* **CreateNode**: создает узел любого типа
* **CreateElement**: создает узел типа XmlDocument
* **CreateAttribute**: создает узел типа XmlAttribute
* **CreateTextNode**: создает узел типа XmlTextNode
* **CreateComment**: создает комментарий

Возьмем xml-документ people.xml из прошлой темы:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?xml version="1.0" encoding="utf-8" ?>  <people>    <person name="Tom">      <company>Microsoft</company>      <age>37</age>    </person>    <person name="Bob">      <company>Google</company>      <age>41</age>    </person>  </people> |

Добавим в этот xml-документ новый элемент <person>:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | using System.Xml;    XmlDocument xDoc = new XmlDocument();  xDoc.Load("people.xml");  XmlElement? xRoot = xDoc.DocumentElement;    // создаем новый элемент person  XmlElement personElem = xDoc.CreateElement("person");    // создаем атрибут name  XmlAttribute nameAttr = xDoc.CreateAttribute("name");    // создаем элементы company и age  XmlElement companyElem = xDoc.CreateElement("company");  XmlElement ageElem = xDoc.CreateElement("age");    // создаем текстовые значения для элементов и атрибута  XmlText nameText = xDoc.CreateTextNode("Mark");  XmlText companyText = xDoc.CreateTextNode("Facebook");  XmlText ageText = xDoc.CreateTextNode("30");    //добавляем узлы  nameAttr.AppendChild(nameText);  companyElem.AppendChild(companyText);  ageElem.AppendChild(ageText);    // добавляем атрибут name  personElem.Attributes.Append(nameAttr);  // добавляем элементы company и age  personElem.AppendChild(companyElem);  personElem.AppendChild(ageElem);  // добавляем в корневой элемент новый элемент person  xRoot?.AppendChild(personElem);  // сохраняем изменения xml-документа в файл  xDoc.Save("people.xml"); |

Добавление элементов происходит по одной схеме. Сначала создаем элемент (xDoc.CreateElement("person")). Если элемент сложный, то есть содержит в себе другие элементы, то создаем эти элементы. Если элемент простой, содержащий внутри себя некоторое текстовое значение, то создаем этот текст (XmlText companyText = xDoc.CreateTextNode("Facebook");).

Затем все элементы добавляются в основной элемент person, а тот добавляется в корневой элемент (xRoot.AppendChild(personElem);).

Чтобы сохранить измененный документ на диск, используем метод Save: xDoc.Save("people.xml")

После этого в xml-файле появится следующий элемент:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <?xml version="1.0" encoding="utf-8"?>  <people>    <person name="Tom">      <company>Microsoft</company>      <age>37</age>    </person>    <person name="Bob">      <company>Google</company>      <age>41</age>    </person>    <people name="Mark">      <company>Facebook</company>      <age>30</age>    </people>  </people> |

### Удаление узлов

Удалим первый узел xml-документа:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | using System.Xml;    XmlDocument xDoc = new XmlDocument();  xDoc.Load("people.xml");  XmlElement? xRoot = xDoc.DocumentElement;  XmlNode? firstNode = xRoot?.FirstChild;  if(firstNode is not null) xRoot?.RemoveChild(firstNode);  xDoc.Save("people.xml"); |

## XPath

**XPath** представляет язык запросов в XML. Он позволяет выбирать элементы, соответствующие определенному селектору.

Рассмотрим некоторые наиболее распространенные селекторы:

.

выбор текущего узла

..

выбор родительского узла

\*

выбор всех дочерних узлов текущего узла

person

выбор всех узлов с определенным именем, в данном случае с именем "person"

@name

выбор атрибута текущего узла, после знака @ указывается название атрибута (в данном случае "name")

@+

выбор всех атрибутов текущего узла

element[3]

выбор определенного дочернего узла по индексу, в данном случае третьего узла

//person

выбор в документе всех узлов с именем "person"

person[@name='Tom']

выбор элементов с определенным значением атрибута. В данном случае выбираются все элементы "person" с атрибутом name='Tom'

person[company='Microsoft']

выбор элементов с определенным значением вложенного элемента. В данном случае выбираются все элементы "person", у которых дочерний элемент "company" имеет значение 'Microsoft'

//person/company

выбор в документе всех узлов с именем "company", которые находятся в элементах "person"

Действие запросов XPath основано на применении двух методов класса **XmlElement**:

* **SelectSingleNode()**: выбор единственного узла из выборки. Если выборка по запросу содержит несколько узлов, то выбирается первый
* **SelectNodes()**: выборка по запросу коллекции узлов в виде объекта XmlNodeList

Для запросов возьмем xml-документ из прошлых тем:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?xml version="1.0" encoding="utf-8" ?>  <people>    <person name="Tom">      <company>Microsoft</company>      <age>37</age>    </person>    <person name="Bob">      <company>Google</company>      <age>41</age>    </person>  </people> |

Теперь выберем все узлы корневого элемента, то есть все элементы person:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | using System.Xml;    XmlDocument xDoc = new XmlDocument();  xDoc.Load("people.xml");  XmlElement? xRoot = xDoc.DocumentElement;    // выбор всех дочерних узлов  XmlNodeList? nodes = xRoot?.SelectNodes("\*");  if (nodes is not null)  {      foreach (XmlNode node in nodes)          Console.WriteLine(node.OuterXml);  } |

Консольный вывод:

<person name="Tom"><company>Microsoft</company><age>37</age></person>

<person name="Bob"><company>Google</company><age>41</age></person>

Выберем все узлы <person>:

|  |  |
| --- | --- |
| 1 | XmlNodeList? personNodes = xRoot?.SelectNodes("person"); |

Выведем на консоль значения атрибутов name у элементов person:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | using System.Xml;    XmlDocument xDoc = new XmlDocument();  xDoc.Load("people.xml");  XmlElement? xRoot = xDoc.DocumentElement;  XmlNodeList? personNodes = xRoot?.SelectNodes("person");  if(personNodes is not null)  {      foreach (XmlNode node in personNodes)          Console.WriteLine(node.SelectSingleNode("@name")?.Value);  } |

Результатом выполнения будет следующий вывод:

Tom

Bob

Выберем узел, у которого атрибут name имеет значение "Tom":

|  |  |
| --- | --- |
| 1  2  3 | XmlNode childnode = xRoot.SelectSingleNode("person[@name='Tom']");  if (childnode != null)      Console.WriteLine(childnode.OuterXml); |

Выберем узел, у которого вложенный элемент "company" имеет значение "Microsoft":

|  |  |
| --- | --- |
| 1  2 | XmlNode? tomNode = xRoot?.SelectSingleNode("person[@name='Tom']");  Console.WriteLine(tomNode?.OuterXml); |

Допустим, нам надо получить только компании. Для этого надо осуществить выборку вниз по иерархии элементов:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | using System.Xml;    XmlDocument xDoc = new XmlDocument();  xDoc.Load("people.xml");  XmlElement? xRoot = xDoc.DocumentElement;    XmlNodeList? companyNodes = xRoot?.SelectNodes("//person/company");  if(companyNodes is not null)  {      foreach (XmlNode node in companyNodes)          Console.WriteLine(node.InnerText);  } |

## Linq to Xml. Создание документа XML

Еще один подход к работе с Xml представляет технология **LINQ to XML**. Вся функциональность LINQ to XML содержится в пространстве имен **System.Xml.Linq**. Рассмотрим основные классы этого пространства имен:

* **XAttribute**: представляет атрибут xml-элемента
* **XComment**: представляет комментарий
* **XDocument**: представляет весь xml-документ
* **XElement**: представляет отдельный xml-элемент

Ключевым классом является XElement, который позволяет получать вложенные элементы и управлять ими. Среди его методов можно отметить следующие:

* **Add()**: добавляет новый атрибут или элемент
* **Attributes()**: возвращает коллекцию атрибутов для данного элемента
* **Elements()**: возвращает все дочерние элементы данного элемента
* **Remove()**: удаляет данный элемент из родительского объекта
* **RemoveAll()**: удаляет все дочерние элементы и атрибуты у данного элемента

Итак, используем функциональность LINQ to XML и создадим новый XML-документ:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | using System.Xml.Linq;    XDocument xdoc = new XDocument();  // создаем первый элемент person  XElement tom = new XElement("person");  // создаем атрибут name  XAttribute tomNameAttr = new XAttribute("name", "Tom");  // создаем два элемента company и age  XElement tomCompanyElem = new XElement("company", "Microsoft");  XElement tomAgeElem = new XElement("age", 37);  // добавляем атрибут и элементы в первый элемент person  tom.Add(tomNameAttr);  tom.Add(tomCompanyElem);  tom.Add(tomAgeElem);    // создаем второй элемент person  XElement bob = new XElement("person");  // создаем для него атрибут name  XAttribute bobNameAttr = new XAttribute("name", "Bob");  // и два элемента - company и age  XElement bobCompanyElem = new XElement("company", "Google");  XElement bobAgeElem = new XElement("age", 41);  bob.Add(bobNameAttr);  bob.Add(bobCompanyElem);  bob.Add(bobAgeElem);  // создаем корневой элемент  XElement people = new XElement("people");  // добавляем два элемента person в корневой элемент  people.Add(tom);  people.Add(bob);  // добавляем корневой элемент в документ  xdoc.Add(people);  //сохраняем документ  xdoc.Save("people.xml");    Console.WriteLine("Data saved"); |

Чтобы создать документ, нам нужно создать объект класса **XDocument**. Это объект самого верхнего уровня в хml-документе.

Элементы создаются с помощью конструктора класса **XElement**. Конструктор имеет ряд перегруженных версий. Первый параметр конструктора передает название элемента, например, person. Второй параметр передает значение этого элемента.

Создание атрибута аналогично созданию элемента. Затем все атрибуты и элементы добавляются в элементы person с помощью метода Add().

Так как документ xml должен иметь один корневой элемент, то затем все элементы person добавляются в один контейнер - элемент people.

В конце корневой элемент добавляется в объект XDocument, и этот объект сохраняется на диске в xml-файл с помощью метода **Save()**.

Если мы откроем сохраненный файл **people.xml**, то увидим в нем следующее содержание:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?xml version="1.0" encoding="utf-8"?>  <people>    <person name="Tom">      <company>Microsoft</company>      <age>37</age>    </person>    <person name="Bob">      <company>Google</company>      <age>41</age>    </person>  </people> |

Конструктор класса XElement позволяют задать набор объектов, которые будут входить в элемент. И предыдущий пример мы могли бы сократить следующим способом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | using System.Xml.Linq;    XDocument xdoc = new XDocument(new XElement("people",      new XElement("person",          new XAttribute("name", "Tom"),          new XElement("company", "Microsoft"),          new XElement("age", 37)),      new XElement("person",          new XAttribute("name", "Bob"),          new XElement("company", "Google"),          new XElement("age", 41))));  xdoc.Save("people2.xml");    Console.WriteLine("Data saved"); |

### Что такое XML?

**XML** (eXtensible Markup Language) — это универсальный расширяемый язык для разметки информации. Он позволяет структурировать, хранить и передавать информацию в определенном формате. Особенность данного языка заключается в том, что самоопределяемым, что позволяет самостоятельно расширять набор условных меток — тегов. Используя их, можно формировать данные в виде древовидной структуры, когда одни элементы содержат в себе другие.

#### Пример XML разметки

В качестве примера можно рассмотреть телефонную книгу, в которой хранятся такие данные как имя человека, номер телефона и заметка.

<?xml version="1.0" encoding="UTF-8"?> <!--XML заголовок-->

<catalog> <!--Корневой элемент-->

<phone> <!--Элемент данных-->

<name>Саша</name> <!--Первое значение данных-->

<number>890953317</number> <!--Второе значение-->

<remark>Не бери трубку!</remark> <!--Третье значение-->

</phone> <!--Закрытие элемента данных-->

<phone>

<name>Дима</name>

<number>890512309</number>

<remark>Босс</remark>

</phone>

<phone group="important"> <!--Используем атрибут для дополнительного свойства-->

<name>Рита</name>

<number>890198735</number>

<remark>Невероятная девчонка</remark>

</phone>

</catalog> <!--Завершение всех данных-->

Первым строкой в файле должен идти заголовок, который указывает какую версию использовать, а также кодировку текста. Затем должен обязательно идти единственных корневой элемент, в который включены все остальные данные. В дальнейшем все данные разбиваются на объекты, включающие другие объекты и так далее, в зависимости от структуры данных. Также у объекта могут быть указаны дополнительные свойства, они называются атрибуты.

### Работа с XML на языке C#

Для взаимодействия с файлом в языке программирования C# имеется несколько отличных библиотек, но для начала мы рассмотрим одну из самых простых реализаций, расположенную в пространстве имен System.XML. Данная библиотека позволяет как читать, так и сохранять данные в файлы.

Реализуем две основные операции, это чтение данных из файла, и сохранение данных в файл. Рассмотрим основной метод приложения.

###### Program.cs

static void Main(string[] args)

{

// Устанавливаем кодировку консоли.

// Нужно только если при использовании англоязычной Windows

// на консоль вместо кириллицы выводятся знаки вопроса (??? ????? ??????)

Console.OutputEncoding = Encoding.Unicode;

// Читаем Xml файл.

ReadXmlFile("example.xml");

// Ждем ввода пользователя.

Console.ReadLine();

// Создаем структуру данных.

var catalog = new Catalog() // Корневой элемент

{

Phones = new List<Phone>() // Коллекция номеров телефонов.

{

new Phone() {Name = "Саша", Number = 890953317, Remark = "Не бери трубку!", Important = false}, // Запись номера телефона.

new Phone() {Name = "Дима", Number = 890512309, Remark = "Босс", Important = false},

new Phone() {Name = "Рита", Number = 890198735, Remark = "Невероятная девчонка", Important = true}

}

};

// Пишем в файл.

WriteXmlFile("result.xml", catalog);

// Сообщаем пользователю о завершении.

Console.WriteLine("ОК");

Console.ReadLine();

}

#### Чтение XML файла на языке C#

Возьмем в качестве исходного файла пример указанный выше, для простоты удалив комментарии.

###### Program.cs

/// <summary>

/// Прочитать Xml файл.

/// </summary>

/// <param name="filename"> Путь к Xml файлу. </param>

private static void ReadXmlFile(string filename)

{

// Создаем экземпляр Xml документа.

var doc = new XmlDocument();

// Загружаем данные из файла.

doc.Load(filename);

// Получаем корневой элемент документа.

var root = doc.DocumentElement;

// Используем метод для рекурсивного обхода документа.

PrintItem(root);

}

/// <summary>

/// Метод для отображения содержимого xml элемента.

/// </summary>

/// <remarks>

/// Получает элемент xml, отображает его имя, затем все атрибуты

/// после этого переходит к зависимым элементам.

/// Отображает зависимые элементы со смещением вправо от начала строки.

/// </remarks>

/// <param name="item"> Элемент Xml. </param>

/// <param name="indent"> Количество отступов от начала строки. </param>

private static void PrintItem(XmlElement item, int indent = 0)

{

// Выводим имя самого элемента.

// new string('\t', indent) - создает строку состоящую из indent табов.

// Это нужно для смещения вправо.

// Пробел справа нужен чтобы атрибуты не прилипали к имени.

Console.Write($"{new string('\t', indent)}{item.LocalName} ");

// Если у элемента есть атрибуты,

// то выводим их поочередно, каждый в квадратных скобках.

foreach(XmlAttribute attr in item.Attributes)

{

Console.Write($"[{attr.InnerText}]");

}

// Если у элемента есть зависимые элементы, то выводим.

foreach(var child in item.ChildNodes)

{

if (child is XmlElement node)

{

// Если зависимый элемент тоже элемент,

// то переходим на новую строку

// и рекурсивно вызываем метод.

// Следующий элемент будет смещен на один отступ вправо.

Console.WriteLine();

PrintItem(node, indent + 1);

}

if(child is XmlText text)

{

// Если зависимый элемент текст,

// то выводим его через тире.

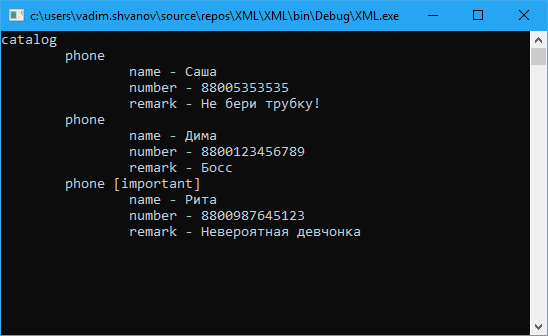
Console.Write($"- {text.InnerText}");

}

}

}

В результате получим следующий вывод на консоль



#### Создание XML файла на языке C#

Для начала нам нужно определить классы модели данных, то есть указать свойства, которые хотим хранить в файле.

###### Phone.cs

namespace XML

{

/// <summary>

/// Телефонный контакт.

/// </summary>

public class Phone

{

/// <summary>

/// Имя.

/// </summary>

public string Name { get; set; }

/// <summary>

/// Телефонный номер.

/// </summary>

public int Number { get; set; }

/// <summary>

/// Заметка.

/// </summary>

public string Remark { get; set; }

/// <summary>

/// Важный контакт.

/// </summary>

public bool Important { get; set; }

/// <summary>

/// Приведение объекта к строке.

/// </summary>

/// <returns> Имя. </returns>

public override string ToString()

{

return Name;

}

}

}

###### Catalog.cs

using System.Collections.Generic;

namespace XML

{

/// <summary>

/// Каталог телефонных номеров.

/// </summary>

public class Catalog

{

/// <summary>

/// Список телефонных номеров.

/// </summary>

public List<Phone> Phones { get; set; } = new List<Phone>();

}

}

Теперь реализуем метод формирования файла на основе имеющихся данных.

###### Program.cs

/// <summary>

/// Сохранить данные в Xml файл.

/// </summary>

/// <param name="filename"> Путь к сохраняемому файлу. </param>

/// <param name="catalog"> Сохраняемые данные. </param>

private static void WriteXmlFile(string filename, Catalog catalog)

{

// Создаем новый Xml документ.

var doc = new XmlDocument();

// Создаем Xml заголовок.

var xmlDeclaration = doc.CreateXmlDeclaration("1.0", "UTF-8", null);

// Добавляем заголовок перед корневым элементом.

doc.AppendChild(xmlDeclaration);

// Создаем Корневой элемент

var root = doc.CreateElement("catalog");

// Получаем все записи телефонной книги.

foreach(var phone in catalog.Phones)

{

// Создаем элемент записи телефонной книги.

var phoneNode = doc.CreateElement("phone");

if (phone.Important)

{

// Если установлен атрибут Важный в true,

// то создаем и добавляем атрибут к элементу записи телефонной книги.

// Создаем атрибут и нужным именем.

var attribute = doc.CreateAttribute("group");

// Устанавливаем содержимое атрибута.

attribute.InnerText = "important";

// Добавляем атрибут к элементу.

phoneNode.Attributes.Append(attribute);

}

// Создаем зависимые элементы.

AddChildNode("name", phone.Name, phoneNode, doc);

AddChildNode("number", phone.Number.ToString(), phoneNode, doc);

AddChildNode("remark", phone.Remark, phoneNode, doc);

// Добавляем запись телефонной книги в каталог.

root.AppendChild(phoneNode);

}

// Добавляем новый корневой элемент в документ.

doc.AppendChild(root);

// Сохраняем документ.

doc.Save(filename);

}

/// <summary>

/// Добавить зависимый элемент с текстом.

/// </summary>

/// <param name="childName"> Имя дочернего элемента. </param>

/// <param name="childText"> Текст, который будет внутри дочернего элемента. </param>

/// <param name="parentNode"> Родительский элемент. </param>

/// <param name="doc"> Xml документ. </param>

private static void AddChildNode(string childName, string childText, XmlElement parentNode, XmlDocument doc)

{

var child = doc.CreateElement(childName);

child.InnerText = childText;

parentNode.AppendChild(child);

}

**XML** является одним из наиболее распространенных стандартов документов, который позволяет в удобной форме сохранять сложные по структуре данные. Поэтому разработчики платформы .NET включили в фреймворк широкие возможности для работы с XML.

XML (eXtensible Markup Language) — расширяемый язык разметки. XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком, с подчёркиванием нацеленности на использование в Интернете.

Рассмотрим, что представляет собой xml-документ и как он может хранить объекты, используемые в программе на C#. Есть следующий класс:

class User

{

    public string Name { get; set; }

    public int Age { get; set; }

    public string Company { get; set; }

}

В программе на C# можно создать список объектов класса User:

User user1 = new User { Name = "Bill Gates", Age = 48, Company = "Microsoft" };

User user2 = new User { Name = "Larry Page", Age = 42, Company = "Google" };

List users = new List { user1, user2 };

Чтобы сохранить список в формате xml можно использовать следующий xml-файл:

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

<users>

  <user name="Bill Gates">

    <company>Microsoft</company>

    <age>48</age>

  </user>

  <user name="Larry Page">

    <company>Google</company>

    <age>48</age>

  </user>

</users>

XML-документ объявляет строка <?xml version=”1.0″ encoding=”utf-8″ ?>. Она задает версию (1.0) и кодировку (utf-8) xml. Далее идет собственно содержимое документа.

XML-документ должен иметь один единственный корневой элемент, внутрь которого помещаются все остальные элементы. В данном случае таким элементом является элемент <users>. Внутри корневого элемента  задан набор элементов <user>. Вне корневого элемента нельзя разместить элементы user.

Каждый элемент определяется с помощью открывающего и закрывающего тегов, например, <user> и </user>, внутри которых помещается значение или содержимое элементов. Также элемент может иметь сокращенное объявление: <user /> – в конце элемента помещается слеш.

Элемент может иметь вложенные элементы и атрибуты. В данном случае каждый элемент user имеет два вложенных элемента company и age и атрибут name.

Названия элементов являются регистрозависимыми, поэтому <company> и <COMPANY> будут представлять разные элементы.

Для работы с XML в C# можно использовать несколько подходов. В первых версиях фреймворка основной функционал работы с XML предоставляло пространство имен System.Xml. В нем определен ряд классов, которые позволяют манипулировать xml-документом:

* XmlNode: представляет узел xml. В качестве узла может использоваться весь документ, так и отдельный элемент;
* XmlDocument: представляет весь xml-документ;
* XmlElement: представляет отдельный элемент. Наследуется от класса XmlNode;
* XmlAttribute: представляет атрибут элемента;
* XmlText: представляет значение элемента в виде текста, то есть тот текст, который находится в элементе между его открывающим и закрывающим тегами;
* XmlComment: представляет комментарий в xml;
* XmlNodeList: используется для работы со списком узлов.

Ключевым классом, который позволяет манипулировать содержимым xml, является **XmlNode**, его основные методы и свойства:

* Свойство Attributes возвращает объект XmlAttributeCollection, который представляет коллекцию атрибутов;
* Свойство ChildNodes возвращает коллекцию дочерних узлов для данного узла;
* Свойство HasChildNodes возвращает true, если текущий узел имеет дочерние узлы;
* Свойство FirstChild возвращает первый дочерний узел;
* Свойство LastChild возвращает последний дочерний узел;
* Свойство InnerText возвращает текстовое значение узла;
* Свойство InnerXml возвращает всю внутреннюю разметку xml узла;
* Свойство Name возвращает название узла. Например, <user> – значение свойства Name равно “user”;
* Свойство ParentNode возвращает родительский узел у текущего узла.

Применим эти классы и их функционал. И вначале для работы с xml создаем новый файл. Называем его **users.xml** и определим в нем следующее содержание:

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

<users>

  <user name="Bill Gates">

    <company>Microsoft</company>

    <age>48</age>

  </user>

  <user name="Larry Page">

    <company>Google</company>

    <age>42</age>

  </user>

</users>

Пройдемся по этому документу и выведем его данные на консоль:

using System.Xml;

class Program

{

    static void Main(string[] args)

    {

        XmlDocument xDoc = new XmlDocument();

        xDoc.Load("users.xml");

        // получим корневой элемент

        XmlElement xRoot = xDoc.DocumentElement;

        // обход всех узлов в корневом элементе

        foreach(XmlNode xnode in xRoot)

        {

            // получаем атрибут name

            if(xnode.Attributes.Count>0)

            {

                XmlNode attr = xnode.Attributes.GetNamedItem("name");

                if (attr!=null)

                    Console.WriteLine(attr.Value);

            }

// обходим все дочерние узлы элемента user

            foreach(XmlNode childnode in xnode.ChildNodes)

            {

                // если узел - company

                if(childnode.Name=="company")

                {

                    Console.WriteLine("Компания: {0}", childnode.InnerText);

                }

                // если узел age

                if (childnode.Name == "age")

                {

                    Console.WriteLine("Возраст: {0}", childnode.InnerText);

                }

            }

            Console.WriteLine();

        }

        Console.Read();

    }

}

Для редактирования xml-документа (изменения, добавления, удаления элементов) можно воспользоваться методами класса **XmlNode**:

* AppendChild: добавляет в конец текущего узла новый дочерний узел;
* InsertAfter: добавляет новый узел после определенного узла;
* InsertBefore: добавляет новый узел до определенного узла;
* RemoveAll: удаляет все дочерние узлы текущего узла;
* RemoveChild: удаляет у текущего узла один дочерний узел и возвращает его.

Класс **XmlElement**, унаследованный от **XmlNode**, добавляет еще ряд методов, которые позволяют создавать новые узлы:

* CreateNode: создает узел любого типа;
* CreateElement: создает узел типа XmlDocument;
* CreateAttribute: создает узел типа XmlAttribute;
* CreateTextNode: создает узел типа XmlTextNode;
* CreateComment: создает комментарий.

Пример:

XmlDocument xDoc = new XmlDocument();

xDoc.Load("users.xml");

XmlElement xRoot = xDoc.DocumentElement;

// создаем новый элемент user

XmlElement userElem = xDoc.CreateElement("user");

// создаем атрибут name

XmlAttribute nameAttr = xDoc.CreateAttribute("name");

// создаем элементы company и age

XmlElement companyElem = xDoc.CreateElement("company");

XmlElement ageElem = xDoc.CreateElement("age");

// создаем текстовые значения для элементов и атрибута

XmlText nameText = xDoc.CreateTextNode("Mark Zuckerberg");

XmlText companyText = xDoc.CreateTextNode("Facebook");

XmlText ageText = xDoc.CreateTextNode("30");