https://metanit.com/sharp/tutorial/16.1.php

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 | class User  {      public string Name { get; set; }      public int Age { get; set; }      public string Company { get; set; }  } |

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

|  |  |
| --- | --- |
| 1  2  3 | User user1 = new User { Name = "Bill Gates", Age = 48, Company = "Microsoft" };  User user2 = new User { Name = "Larry Page", Age = 42, Company = "Google" };  List<User> users = new List<User> { user1, user2 }; |

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" ?>  <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 <users> задан набор элементов <user>. Вне корневого элемента мы не можем 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.

Атрибуты определяются в теле элемента и имеют следующую форму: название="значение". Например, <user 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 <users>, каждый объект User - с элементом <user>, а каждое свойство объекта User - с атрибутом или вложенным элементом элемента <user>

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" ?>  <users>    <user>      <name>Bill Gates</name>      <company>Microsoft</company>      <age>48</age>    </user>    <user>      <name>Larry Page</name>      <company>Google</company>      <age>48</age>    </user>  </users> |

Теперь рассмотрим основные подходы для работы с XML, которые имеются в C#.

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 *users.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" ?>  <users>    <user name="Bill Gates">      <company>Microsoft</company>      <age>48</age>    </user>    <user name="Larry Page">      <company>Google</company>      <age>42</age>    </user>  </users> |

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("D:\\users.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($"Компания: {childnode.InnerText}");              }              // если узел age              if (childnode.Name == "age")              {                  Console.WriteLine($"Возраст: {childnode.InnerText}");              }          }          Console.WriteLine();      }  } |

В итоге я получу следующий вывод на консоли:

Bill Gates

Компания: Microsoft

Возраст: 48

Larry Page

Компания: Google

Возраст: 42

Чтобы начать работу с документом xml, нам надо создать объект XmlDocument и затем загрузить в него xml-файл: xDoc.Load("users.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 <user>, 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 | using System.Xml;    List<User> users = new List<User>();    XmlDocument xDoc = new XmlDocument();  xDoc.Load("D:\\users.xml");  XmlElement? xRoot = xDoc.DocumentElement;  if (xRoot != null)  {      foreach (XmlElement xnode in xRoot)      {          User user = new User();          XmlNode? attr = xnode.Attributes.GetNamedItem("name");          user.Name = attr?.Value;            foreach (XmlNode childnode in xnode.ChildNodes)          {              if (childnode.Name == "company")                  user.Company = childnode.InnerText;                if (childnode.Name == "age")                  user.Age = int.Parse(childnode.InnerText);          }          users.Add(user);      }      foreach (User u in users)          Console.WriteLine($"{u.Name} ({u.Company}) - {u.Age}");  }    class User  {      public string? Name { get; set; }      public int Age { get; set; }      public string? Company { get; set; }  } |

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

Консольный вывод программы:

Bill Gates (Microsoft) - 48

Larry Page (Google) -42

**Сериализация в XML. XmlSerializer**

<https://metanit.com/sharp/tutorial/6.4.php>

Для удобного сохранения и извлечения объектов из файлов xml может использоваться класс **XmlSerializer**.

Во-первых, XmlSerializer предполагает некоторые ограничения. Например, класс, подлежащий сериализации, должен иметь стандартный конструктор без параметров. Также сериализации подлежат только открытые члены. Если в классе есть поля или свойства с модификатором private, то при сериализации они будут игнорироваться.

Во-вторых, XmlSerializer требует указания типа:

|  |  |
| --- | --- |
| 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  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | using System;  using System.IO;  using System.Xml.Serialization;    namespace Serialization  {      // класс и его члены объявлены как public      [Serializable]      public class Person      {          public string Name { get; set; }          public int Age { get; set; }            // стандартный конструктор без параметров          public Person()          { }            public Person(string name, int age)          {              Name = name;              Age = age;          }      }      class Program      {          static void Main(string[] args)          {              // объект для сериализации              Person person = new Person("Tom", 29);              Console.WriteLine("Объект создан");                // передаем в конструктор тип класса              XmlSerializer formatter = new XmlSerializer(typeof(Person));                // получаем поток, куда будем записывать сериализованный объект              using (FileStream fs = new FileStream("persons.xml", FileMode.OpenOrCreate))              {                 formatter.Serialize(fs, person);                    Console.WriteLine("Объект сериализован");              }                // десериализация              using (FileStream fs = new FileStream("persons.xml", FileMode.OpenOrCreate))              {                  Person newPerson = (Person)formatter.Deserialize(fs);                    Console.WriteLine("Объект десериализован");                  Console.WriteLine($"Имя: {newPerson.Name} --- Возраст: {newPerson.Age}");              }                Console.ReadLine();          }      }  } |

Итак, класс Person общедоступный и имеет общедоступные свойства, поэтому он может сериализоваться. При создании объекта XmlSerializer передаем в конструктор тип класса. Метод Serialize добавляет данные в файл *persons.xml*. А метод Deserialize извлекает их оттуда.

Если мы откроем файл *persons.xml*, то увидим содержание нашего объекта:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <?xml version="1.0"?>  <Person xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xmlns:xsd="<http://www.w3.org/2001/XMLSchema>">    <Name>Tom</Name>    <Age>29</Age>  </Person> |

Равным образом мы можем сериализовать массив или коллекцию объектов, но главное требование состоит в том, чтобы в них был определен стандартный конструктор:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | Person person1 = new Person("Tom", 29);  Person person2 = new Person("Bill", 25);  Person[] people = new Person[] { person1, person2 };    XmlSerializer formatter = new XmlSerializer(typeof(Person[]));    using (FileStream fs = new FileStream("people.xml", FileMode.OpenOrCreate))  {      formatter.Serialize(fs, people);  }    using (FileStream fs = new FileStream("people.xml", FileMode.OpenOrCreate))  {      Person[] newpeople = (Person[])formatter.Deserialize(fs);        foreach (Person p in newpeople)      {          Console.WriteLine($"Имя: {p.Name} --- Возраст: {p.Age}");      }  } |

Но это был простой объект. Однако с более сложными по составу объектами работать так же просто. Например:

|  |  |
| --- | --- |
| 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  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65 | using System;  using System.IO;  using System.Xml.Serialization;    namespace Serialization  {      [Serializable]      public class Person      {          public string Name { get; set; }          public int Age { get; set; }          public Company Company { get; set; }            public Person()          { }            public Person(string name, int age, Company comp)          {              Name = name;              Age = age;              Company = comp;          }      }        [Serializable]      public class Company      {          public string Name { get; set; }            // стандартный конструктор без параметров          public Company() { }            public Company(string name)          {              Name = name;          }      }      class Program      {          static void Main(string[] args)          {              Person person1 = new Person("Tom", 29, new Company("Microsoft"));              Person person2 = new Person("Bill", 25, new Company("Apple"));              Person[] people = new Person[] { person1, person2 };                XmlSerializer formatter = new XmlSerializer(typeof(Person[]));                using (FileStream fs = new FileStream("people.xml", FileMode.OpenOrCreate))              {                 formatter.Serialize(fs, people);              }                using (FileStream fs = new FileStream("people.xml", FileMode.OpenOrCreate))              {                  Person[] newpeople = (Person[])formatter.Deserialize(fs);                    foreach (Person p in newpeople)                  {                      Console.WriteLine($"Имя: {p.Name} --- Возраст: {p.Age} --- Компания: {p.Company.Name}");                  }              }              Console.ReadLine();          }      }  } |

Класс Person содержит свойство Company, которое будет хранить объект класса Company. Члены класса Company объявляются с модификатором public, кроме того также присутствует стандартный конструктор без параметров. В итоге после сериализации мы получим следующий xml-документ:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <?xml version="1.0"?>  <ArrayOfPerson xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xmlns:xsd="<http://www.w3.org/2001/XMLSchema>">    <Person>      <Name>Tom</Name>      <Age>29</Age>      <Company>        <Name>Microsoft</Name>      </Company>    </Person>    <Person>      <Name>Bill</Name>      <Age>25</Age>      <Company>        <Name>Apple</Name>      </Company>    </Person>  </ArrayOfPerson> |