Сериализация данных с помощью стандартных средств платформы .NET

Исходное описание данных:

```
public class Faculty
{
    public List<Student> Students = new List<Student>();
    public List<Teacher> Teachers = new List<Teacher>();
}

public class Student
{
    public const string XmlName = "Student";
    public int? ID;
    public string FirstName;
    public string LastName;
    public Student () { }
}

public class Teacher
{
    public const string XmlName = "Teacher";
    public string Name;
    public string Subject;
}
```

Инициализация данных:

```
    Faculty faculty = new Faculty();
    faculty.Students.Add(
        new Student() { FirstName = "Igor", LastName = "Ivanov", ID = 1 });
    faculty.Students.Add(
        new Student() { FirstName = "Andrew", LastName = "Petrov", ID = 2 });
    faculty.Students.Add(
        new Student() { FirstName = "Sergey", LastName = "Volkov", ID = 3 });
    faculty.Teachers.Add(
        new Teacher() { Name = "Andreev", Subject = "Math"});
    faculty.Teachers.Add(
        new Teacher() { Name = "Dmitriev", Subject = "Physics" });
    faculty.Teachers.Add(
        new Teacher() { Name = "Alekseev", Subject = "Programming" }) ;
```

Бинарная сериализация

Для бинарной сериализации типы данных должны быть помечены атрибутом [Serializable]:

```
[Serializable]
public class Faculty
{
    public List<Student> Students = new List<Student>();
    public List<Teacher> Teachers = new List<Teacher>();
}
..
static void BinSer(Faculty f)
{
    BinaryFormatter fmt = new BinaryFormatter();
    using (var str =
        new FileStream(@"C:\faculty5.bin", FileMode.Create))
        fmt.Serialize(str, f);
```

```
}
```

Сериализация в JSON-формате

Формат JSON представляет собой более компактное размещение иерархических данных.

```
static void JSonSer(Faculty f)
{
    DataContractJsonSerializer ser = new DataContractJsonSerializer(typeof(Faculty));
    using(var stream = new FileStream(@"C:\faculty4.json", FileMode.Create))
        ser.WriteObject(stream, f);
}
```

Данные в JSON-формате:


```
{"Students":[{"FirstName":"Igor","ID":1,"LastName":"Ivanov"},{"FirstName":"Andrew","ID":2
,"LastName":"Petrov"},{"FirstName":"Sergey","ID":3,"LastName":"Volkov"}],"Teachers":[{"Co
ntactData":{"m_Item1":"email","m_Item2":"some.ru"},"Name":"Andreev","Subject":"Math"},{"C
ontactData":{"m_Item1":"address","m_Item2":"Spb"},"Name":"Dmitriev","Subject":"Physics"},
{"ContactData":{"m_Item1":"address","m_Item2":"Spb"},"Name":"Alekseev","Subject":"Program
ming"}]}
```


Стандартная сериализация в XML-формате

Обязательные требования: сериализуемые классы открыты, конструктор без параметров, сериализуемые поля и свойства поддерживаются в XmlSerializer.

```
static void StandardXml(Faculty f)
{
    XmlSerializer ser = new XmlSerializer(typeof(Faculty));
    TextWriter stream = new StreamWriter(@"C:\faculty2.xml");
    ser.Serialize(stream, f);
    stream.Close();
}
```

Данные в XML-формате

```
<?xml version="1.0" encoding="utf-8"?>
<Faculty xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" .. >
  <Students>
    <Student>
      <ID>1</ID>
      <FirstName>Igor</FirstName>
      <LastName>Ivanov</LastName>
    </Student>
    <Student>
      <ID>2</ID>
      <FirstName>Andrew</FirstName>
      <LastName>Petrov</LastName>
    </Student>
    <Student>
      <ID>3</ID>
      <FirstName>Sergey</FirstName>
      <LastName>Volkov</LastName>
    </Student>
  </Students>
  <Teachers>
    <Teacher>
      <Name>Andreev</Name>
      <Subject>Math</Subject>
```

```
    </Teacher>
    <Teacher>
      <Name>Dmitriev</Name>
      <Subject>Physics</Subject>
    </Teacher>
    <Teacher>
      <Name>Alekseev</Name>
      <Subject>Programming</Subject>
    </Teacher>
  </Teachers>
</Faculty>
```

С помощью атрибутов [XmlIgnore], [XmlAttribute], [XmlElement] можно настраивать структуру Xml-файла: игнорировать свойства или поля, оформлять в виде Xml-атрибутов, задавать имена Xml-элементов. Если в классе используются неподдерживаемые типы (Tuple, Dictionary и др.), то сериализация не будет выполнена. Для решения проблемы можно использовать дополнительные свойства, которые будут использоваться только для сериализации:

```
public class Teacher
{
    public const string XmlName = "Teacher";
    [XmlAttribute(“FullName”)]
    public string Name;
    [XmlIgnore]
    public string Subject;
    [XmlIgnore]
    public Tuple<string, string> ContactData;
    [XmlElement(“Contacts”)]
    public string XmlContactData
    {
        get
        {
            return ContactData.Item1 + ": " + ContactData.Item2;
        }
        set
        {
            if (value.Length > 0)
            {
                string[] ar = value.Split(new string[] { ": " },
                    StringSplitOptions.None);
                ContactData = Tuple.Create(ar[0], ar[1]);
            }
        }
    }
}
```

В этом случае структура фрагмента XML-файла, относящегося к объектам Teacher, будет следующей:

```
<Teachers>
    <Teacher FullName="Andreev">
      <Contacts>email: some.ru</Contacts>
    </Teacher>
    <Teacher FullName="Dmitriev">
      <Contacts>address: Spb</Contacts>
    </Teacher>
    <Teacher FullName="Alekseev">
      <Contacts>address: Spb</Contacts>
    </Teacher>
```

```
</Teachers>
```

Стандартная сериализация в SOAP-формате

Сериализация SOAP не поддерживает обобщенные типы данных, в том числе списки List<T>. Поэтому необходимо перейти к ArrayList. В следующем фрагменте рассматривается сохранение только одного списка Teachers:

```
static void SoapSer(Faculty f)
{
    SoapFormatter fmt = new SoapFormatter();
    using (Stream stream = new FileStream("faculty3.soap", FileMode.Create))
    {
        System.Collections.ArrayList ar = new System.Collections.ArrayList();
        f.Teachers.ForEach(t => ar.Add(t));
        fmt.Serialize(stream, ar);
    }
}
```

```
<SOAP-ENV:Envelope xmlns:xsi= .. >
<SOAP-ENV:Body>
<a1:ArrayList id="ref-1"
xmlns:a1="http://schemas.microsoft.com/clr/ns/System.Collections">
<_items href="#ref-2"/>
<_size>3</_size>
<_version>3</_version>
</a1:ArrayList>
<SOAP-ENC:Array id="ref-2" SOAP-ENC:arrayType="xsd:anyType[4]">
<item href="#ref-3"/>
<item href="#ref-4"/>
<item href="#ref-5"/>
</SOAP-ENC:Array>
<a3:Teacher id="ref-3"
xmlns:a3="http://schemas.microsoft.com/clr/nsassem/SerReaderWriter/SerReaderWriter%2C%20V
ersion%3D1.0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
<Name id="ref-7">Andreev</Name>
<Subject id="ref-8">Math</Subject>
</a3:Teacher>
<a3:Teacher id="ref-4"
xmlns:a3="http://schemas.microsoft.com/clr/nsassem/SerReaderWriter/SerReaderWriter%2C%20V
ersion%3D1.0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
<Name id="ref-9">Dmitriev</Name>
<Subject id="ref-10">Physics</Subject>
</a3:Teacher>
<a3:Teacher id="ref-5"
xmlns:a3="http://schemas.microsoft.com/clr/nsassem/SerReaderWriter/SerReaderWriter%2C%20V
ersion%3D1.0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
<Name id="ref-11">Alekseev</Name>
<Subject id="ref-12">Programming</Subject>
</a3:Teacher>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Формирование XML-файла с помощью LINQ to XML

Технология Linq to Xml обеспечивает удобную работу с Xml-форматом с помощью типовых запросов. Следующий фрагмент сохраняет данные, конструируя структуру файла с помощью вложенных Linq-запросов:

```
static void LinqToXml(Faculty f)
{
    XElement x = new XElement("Faculty",
                    from st in f.Students
                    select new XElement("Student",
                        new XAttribute("ID", st.ID),
                        new XElement("LastName", st.LastName),
                        new XElement("FirstName", st.FirstName)),
                    from t in f.Teachers
                    select new XElement("Teacher",
                        new XElement("Name", t.Name),
                        new XElement("Subject", t.Subject)
                        )
                    );
    Console.WriteLine(x.ToString());
    Console.ReadKey();
}
```