Лабораторная работа JSON сериализация

Существует два основных способа преобразования (сериализации) объектов .NET в JSON:

- Использование класса DataContractJsonSerializer;
- Использование класса JavaScriptSerializer.

Использование класса DataContractJsonSerializer

Класс DataContractJsonSerializer сериализует объекты .NET в поток (Stream) в виде текста в формате JSON или XML сопоставимый с JSON.

Для того чтобы сериализация объекта стала возможна он должен быть отмечен атрибутом DataContract, а его члены подлежащие сериализации атрибутом DataMember. Чтобы эти атрибуты и сам класс DataContractJsonSerializer стали доступны, необходимо подключить к проекту сборку System.Runtime.Serialize.

В качестве примера создадим класс, который описывает организацию.

```
[DataContract]
class Company
{
   [DataMember]
   public string Name { get; set; }
   [DataMember]
   public string INN { get; set; }
   [DataMember]
   public string Adress { get; set; }
}
```

Создадим объект класса DataContractJsonSerializer для сериализации объектов класса Company.

```
DataContractJsonSerializer serializer = new
DataContractJsonSerializer(typeof(Company));
```

Обратите внимание. Сериализуемый тип передаётся в качестве параметра конструктора.

Далее необходимо создать поток для сериализованных данных.

```
MemoryStream stream = new MemoryStream();
```

Или XmlWriter для XML сопоставимого с JSON. Например:

```
XmlWriter writer = new XmlTextWriter("test.xml", null);
```

Сам процесс сериализации осуществляется с помощью метода WriteObject класса DataContractJsonSerializer. В качестве первого параметра ему передаётся либо поток, либо объект XmlWriter. В качестве второго параметра сериализуемый объект.

В итоге получается или поток с текстом в формате JSON.

```
serializer.WriteObject(stream, company);
```

Или XML сопоставимый с JSON.

```
serializer.WriteObject(writer, company);
writer.Close();
```

Использование класса JavaScriptSerializer

Класс JavaScriptSerializer расположен в пространстве имён System.Web.Script.Serialization (требует подключения сборки System.Web.Extensions).

Его отличительная особенность в том, что он сериализует объекты в JSON возвращая обычную строку (string) с текстом в формате JSON. При этом совершенно не требуется обозначать сериализуемый объект или его члены специальными атрибутами.

Конструктор класса JavaScriptSerializer не требует передачи каких либо параметров, а для выполнения сериализации объект передаётся в качестве единственного параметра методу Serialize.

```
JavaScriptSerializer serializer = new JavaScriptSerializer();
string json = serializer.Serialize(company);
```

Десериализация из JSON

Данные сериализованные в JSON можно получить обратно.

Использование класса DataContractJsonSerializer

Десериализация производится при помощи метода ReadObject. В качестве единственного параметра ему передаётся поток с текстом в формате JSON или объект XmlReader для чтения XML сопоставимого с JSON.

При десериализации из потока необходимо установить указатель на позицию соответствующую началу данных в формате JSON (в простейшем случае это 0)..

```
stream.Position = 0;
Company company = (Company)serializer.ReadObject(stream);
```

Если требуется десериализовать данные их XML сопоставимого с JSON, создаём XmlReader для этого XML (в данном примере XML хранится в файле) и передаём его в метод ReadObject класса DataContractJsonSerializer.

```
XmlReader reader = new XmlTextReader("test.xml");
Company company = (Company)serializer.ReadObject(reader);
```

Использование класса JavaScriptSerializer

Для десериализации у класса JavaScriptSerializer есть два метода:

- **Deserialize**.
  Обобщённый метод. Десериализует строку в формате JSON в заданный .NET объект;
- **DeserializeObject**.
  Десериализует строку в формате JSON в словарь в формате Dictionary <string, object>.

Пример использования метода Deserialize:

```
Company company = serializer.Deserialize<Company>(json);
```

Пример использования метода DeserializeObject:

```
Dictionary<string,object> company = (Dictionary < string,object>)
serializer.DeserializeObject(json);
```

Метод Deserialize выполняет простую десериализацию, восстанавливая из JSON исходный объект. Метод DeserializeObject возвращает коллекцию, структура которой соответствует структуре исходного JSON. Что даёт возможность работать с JSON как с обычной коллекцией C# извлекая из неё для последующей обработки лишь необходимые данные.

```
string Adress = company["Adress"].ToString();
```

Таким образом, в зависимости от требований задачи, можно выбрать наиболее подходящее средство для работы с JSON.

Подключить ссылки на сборки System.Runtime.Serialization

System.Web.Extensions

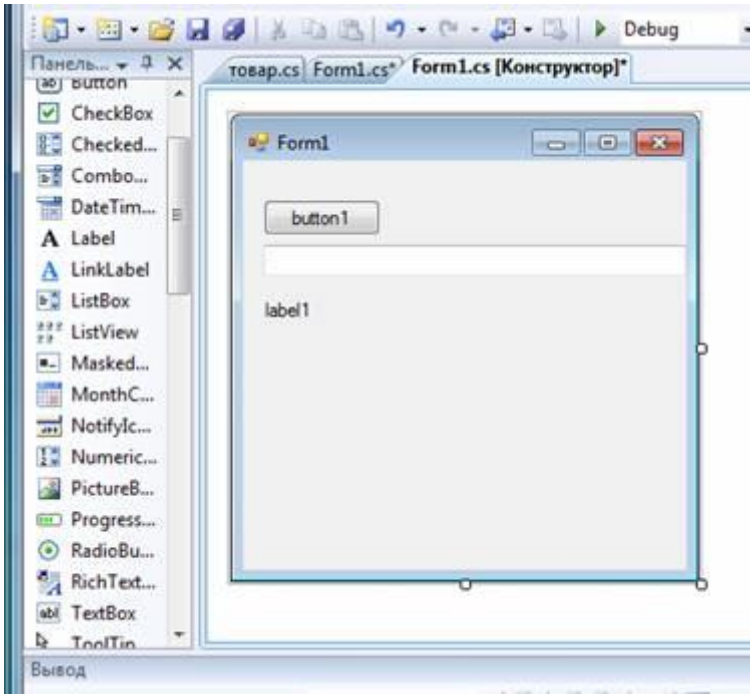
```
using System;
using System.Runtime.Serialization;
using System.Web.Script.Serialization ;

namespace json2
{
    [DataContract]
    class Company
    {
        [DataMember]
        public string Name { get; set; }
        [DataMember]
        public string INN { get; set; }
        [DataMember]
        public string Adress { get; set; }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Company company = new Company();
            company.INN = "1";
            company.Name = "Romashka";
            company.Adress = "Barnaul";
            JavaScriptSerializer serializer = new
JavaScriptSerializer();
            string json = serializer.Serialize(company);
            Console.WriteLine(json);
            Console.ReadLine();
        }
    }
}
```

JSON парсинг в Net

Приложение, демонстрирующее парсинг объектов в нотацию JSON

1) Создайте приложение Windows Form, для поддержки функциональности сериализации включите в ссылки пространство имен System.Runtime.Serialization, System.ServiceModel.Web



на форме разместите объекты: командную кнопку (button1), текстовое поле (textBox1) и надпись (label)

2) Добавьте в приложение класс «товар» с полями «код» - int, «наименование» string и «цена» double.

3) Добавьте в класс конструктор с тремя параметрами для инициализации полей объекта

код конструктора:

```
public товар(int kod, string naimen, double cena)
        {
            this.код = kod;
            this.наименование = naimen;
```

```
            this.цена = cena;
        }
```

4) Добавьте в определении класса разметку для сериализации

`[System.Runtime.Serialization.DataContract]` для класса и

`[System.Runtime.Serialization.DataMember]` для каждого поля класса.

Конструктор сериализовать не будем.

5) На событие нажатие кнопки добавим код создания экземпляра класса «товар»

```
    товар tovar1 = new товар(10, "Стол", 4500);
```

и распечатаем значения полей созданного объекта

```
    this.label1.Text = string.Format("код={0}\n наименование={1} \n
цена={2}", tovar1.код,tovar1.наименование,tovar1.цена  );
```

Запустите приложение, убедитесь, что объект создался и значения его полей правильно инициализировались при создании. Далее измените значения его полей и выведите новые значения в label1 (добавьте текст к предыдущему)

6) Выполните JSON сериализацию объекта при помощи экземпляра объекта серализатора

```
System.Runtime.Serialization.Json.DataContractJsonSerializer ps
= new System.Runtime.Serialization.Json.DataContractJsonSerializer(typeof ( т
овар));
```

Здесь ps – объект сериализатор, провайдер сериализации для класса «товар»

При создании этого объекта анализируется класс и в соответствии с разметкой настраивается сериализатор.

Далее методом `WriteObject`

```
ps.WriteObject(память, tovar1);
```

происходит запись экземпляра класса «товар» - объекта tovar1 в поток памяти – объект «память». Переменная объектного типа «память» должна быть создана предварительно

```
System.IO.Stream память = new System.IO.MemoryStream();
```

Далее, из потока памяти «память» при помощи
объекта StreamReader «чение_из_памяти» происходит считывание текста результата сериализации объекта в строку s.

```
память.Position =0;
System.IO.StreamReader чтение_из_памяти = new System.IO.StreamReader(память);
string s=чтение_из_памяти.ReadToEnd();
```

и строка выводится в текстовое поле

```
this.textBox1.Text = s;
```

7) Проведем процедуру десериализации строки в объект

Запишем строку s1 в нотации JSON, при этом, естественно двойные кавычки приходится маскировать символом “\”

```
string s1 = "{\"код\":11,\"наименование\":\"Стул\",\"цена\":500}";
```

Сохраняем строку в поток памяти

```
System.IO.Stream поток
= new System.IO.MemoryStream(Encoding.UTF8.GetBytes(s1));
```

В надпись label1 для контроля выведем размер потока

```
        this.label1.Text+=string.Format("\n n={0}",    поток.Length );
```

Далее выполним процедуру десериализации при помощи созданного ранее объекта сериализатора или
провайдера сериализации – ps. Десериализация выполняется
методом ReadObject

```
  товар tov2 = (товар)ps.ReadObject(поток);
```

Обратите внимание на явное приведение к типу «товар» результата выполнения метода readObject.

И, наконец, для контроля выводим в label1 значения полей полученного в результате десериализации объекта tovar2

```
        this.label1.Text += string.Format("\n код={0}\n наименование={1} \n
цена={2}", tov2.код, tov2.наименование, tov2.цена);
```

****

Код класса «товар»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace JSON
{
 [System.Runtime.Serialization.DataContract]  class товар
    {
 [System.Runtime.Serialization.DataMember]
        public int код;
        [System.Runtime.Serialization.DataMember]
        public string наименование;
        [System.Runtime.Serialization.DataMember]
        public double цена;

        public товар(int kod, string naimen, double cena)
        {
            this.код = kod;
            this.наименование = naimen;
            this.цена = cena;
        }
    }
}
```

****

Код формы приложения

```
using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;


namespace JSON

{


    public partial class Form1 : Form

    {

        public Form1()

        {

            InitializeComponent();

        }


        private void button1_Click(object sender, EventArgs e)

        {

            товар tovar1 = new товар(10, "Стол", 4500);


            this.label1.Text = string.Format("код={0}\n наименование={1} \n
цена={2}", tovar1.код,tovar1.наименование,tovar1.цена  );


            System.Runtime.Serialization.Json.DataContractJsonSerializer ps
= new System.Runtime.Serialization.Json.DataContractJsonSerializer(typeof ( т
овар));
```

```
            System.IO.Stream память = new System.IO.MemoryStream();


            ps.WriteObject(память, tovar1);

            память.Position =0;

            System.IO.StreamReader чтение_из_памяти = new System.IO.StreamRea
der(память);


            string s=чтение_из_памяти.ReadToEnd();

            this.textBox1.Text = s;




// Десериализация строки в объект


            string s1
= "{\"код\":11,\"наименование\":\"Стул\",\"цена\":500}";


            System.IO.Stream поток = new System.IO.MemoryStream(Encoding.UTF8
.GetBytes(s1));

         this.label1.Text+=string.Format("\n n={0}",    поток.Length );


         товар tov2 = (товар)ps.ReadObject(поток);

         this.label1.Text += string.Format("\n код={0}\n наименование={1}
\n цена={2}", tov2.код, tov2.наименование, tov2.цена);

        }

    }


}
```